BIBLIOTECA BASICA INFORMATICA



INGELEK

BIBLIOTECA BASICA INFORMATICA

LOGO



INGELEK

Director editor:
Antonio M. Ferrer Abelló.

Director de producción:
Vicente Robles.

Coordinador y supervisión técnica:
Enrique Monsalve.

Colaboradores:
Angel Segado
Casimiro Zaragoza
Fernando Ruíz
Francisco Ruíz
Jesús Pedraza
Juanjo Alba Ríos
Margarita Caffaratto
María Angeles Gálvez
Marina Caffaratto
Masé González Balandín
Patricia Mordini

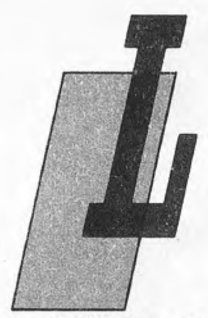
Diseño:
Bravo/Lofish.

Dibujos:
José Ochoa.

ISBN del tomo: 84-85831-52-7
ISBN de la obra: 84-85831-31-4
Fotocomposición: Pérez Díaz, S. A.
Imprime: Héroes, S. A.
Depósito Legal: M-4865-1986
Precio en Canarias, Ceuta y Melilla: 380 pts.

INDICE

PROLOGO



ogo es sinónimo de sencillez y potencia, dos características que rinden honor a los autores del lenguaje y que hacen que éste sea el mejor candidato a la utilización, siempre más cercana, de la informática en la escuela.

La finalidad de este libro de la B.B.I. es guiar al lector en el aprendizaje de las informaciones necesarias para poderse enfrentar después con problemas más complejos. Quien ya posea conocimientos en este campo encontrará en el texto informaciones interesantes.

Desde luego, este libro, como otros, no podrá enseñar completamente lo que quiere decir "saber programar bien"; sólo la experiencia conseguida por cada uno de nosotros y completada por un buen bagaje teórico convierten a un programador cualquiera en un programador competente.

En el libro se tratarán las versiones del lenguaje implementadas en los sistemas COMMODORE 64 y APPLE II; por motivos de claridad, sin embargo, los primeros capítulos que examinan el lenguaje desde el punto de vista general, basarán sus ejemplos en el LOGO TERRPIN del CMB 64. En los capítulos sucesivos se analizarán los detalles técnicos relativos a cada versión.

CAPITULO I

LA TORTUGA, SU RASTRO Y LA RESOLUCION GRAFICA

El símbolo que, además de ser una novedad, ha contribuido a dar fama en el mundo de la informática al lenguaje "LOGO" es la tortuga ("turtle"): moviéndola en la pantalla va dejando una pista a lo largo de su recorrido, dibujando imágenes de diferentes tipos y colores.

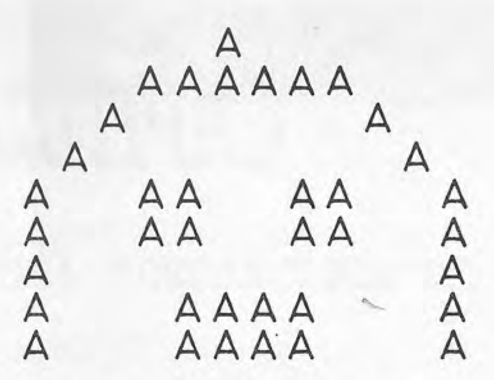
Estudiaremos en este capítulo cuáles son las funciones de la tortuga y, en particular, las instrucciones que están en la base de sus movimientos y del dibujo en general.

Control directo de la tortuga

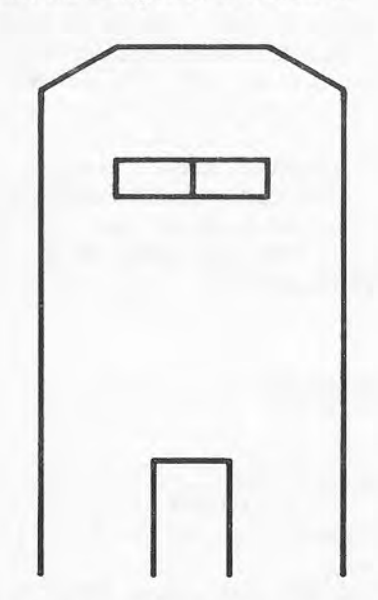
Nuestro ordenador puede trabajar de dos formas diferentes a la hora de visualizar una imagen: en "BAJA" y en "ALTA" resolución gráfica (hablaremos también del modo TEXTO y del modo GRAFICO).

El primer tipo de visualización es típico de la computadora en el momento del encendido; si pulsamos las teclas de la consola veremos inmediatamente en la pantalla los caracteres correspondientes.

Si queremos realizar en la pantalla imágenes a nuestro gusto utilizando exclusivamente los caracteres, podemos manejarlos de tal manera que obtengamos una figura que se aproxime a la que deseamos:



Al contrario, el modo de "alta resolución" da la posibilidad al programador de trazar gráficos y figuras con mayor precisión: él tiene a su disposición una pantalla gráfica en la cual puede dibujar, *punto por punto,* las imágenes necesarias.



Por lo general, la utilización del modo de alta resolución presenta notables dificultades de utilización por parte del usuario: el lenguaje LOGO ha sido estudiado (también) para permitir al programador controlar individualmente cada punto de la pantalla de manera fácil y eficaz, permitiéndole aprovechar completamente las posibilidades gráficas de la computadora sin recurrir a pesados, además de difíciles, cálculos de coordenadas.

El LOGO permite la completa (luego veremos qué quiere decir este término) utilización de una ventana vídeo formada por "X" puntos horizontales y por "Y" verticales (dos parámetros variables según el tipo de ordenador utilizado) sobre la cual se pueden trazar gráficos y dibujos.

La primera instrucción que vamos a examinar será, considerando las premisas:

DRAW (return)

que borra la pantalla, pone en el centro a la tortuga y predispone la pantalla en ALTA RESOLUCION. En particular, la parte superior de la pantalla visualiza la página gráfica, mientras la inferior queda de manera texto, es decir, queda en baja resolución con el fin de poder seguir nuevas instrucciones y, eventualmente, permitir la lectura de los mensajes de error.

Esta es una de las tres combinaciones posibles de pantalla que pueden ser elegidas por medio de las teclas de funciones:

F1: Pantalla completa en forma texto.
F3: Pantalla dividida en alta resolución y forma texto.
F5: Pantalla completamente en alta resolución.

En los ordenadores no dotados de teclas de función y en los que las tienen es posible conseguir también los mismos resultados pulsando:

TEXTSCREEN (return): forma texto.
SPLISCREEN (return): gráfica y texto simultáneamente.
FULLSCREEN (return): sólo gráfica.

Pulsamos ahora SPLITSCREEN (return) y podremos observar a la vez los mandos pulsados (en la parte baja de la pantalla) y los resultados relativos (en la parte superior), como indica la Figura 1.

Escribamos, por ejemplo:

FORWARD 50 <RETURN>

Veremos cómo la tortuga, desde su posición inmóvil en el centro de la pantalla, se mueve hacia delante 50 posiciones, trazando una línea recta entre el punto donde anteriormente se encontraba y el punto alcanzado (Fig. 2).

Por lo tanto, la sintaxis es (en lo sucesivo no explicitaremos ya la necesidad de pulsar la tecla <RETURN> de retorno de carro):

FORWARD (Posiciones a avanzar)

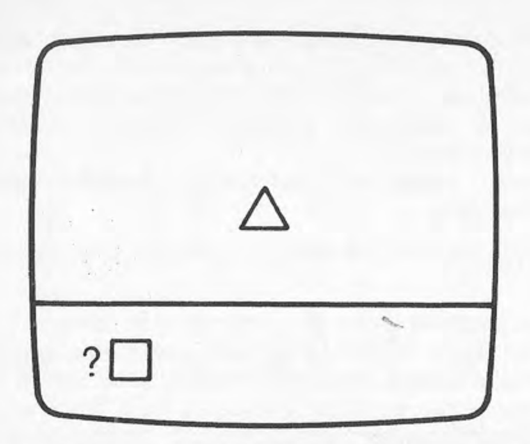


Figura 1.—Al pulsar SPLITSCREEN la parte superior de la pantalla queda en modo gráfico y la parte inferior en modo texto.

por tanto, pulsando:

FORWARD 10 Avanza hacia adelante 10 posiciones
FORWARD 100 Avanza hacia adelante 100 posiciones
FORWARD 120 Avanza hacia adelante 120 posiciones

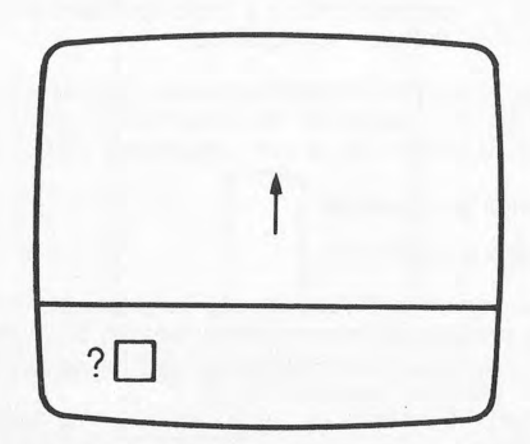


Figura 2.—El comando FORWARD permite que la tortuga avance en línea recta.

Si ahora queremos obtener el resultado opuesto, haciendo retroceder el cursor escribiremos:

BACK 50

y la tortuga, si nos hemos quedado en la figura 1.2, se vuelve a colocar como en la figura 1.1.

La sintaxis es, por consiguiente, similar a la adoptada por FORWARD:

BACK (Posiciones a retroceder)

Pulsando:

BACK 10 Retrocede 10 posiciones
BACK 100 Retrocede 100 posiciones
BACK 120 Retrocede 120 posiciones

Ahora que sabemos cómo movernos en línea recta, veamos cómo podemos variar nuestra trayectoria.

Para hacer esto usamos dos instrucciones cuyo nombre explica ya ampliamente su función: RIGHT (derecha) y LEFT (izquierda).

Pulsamos:

RIGHT 90

veremos el cursor (la tortuga) girar hacia la derecha 90 grados.

Y con:

LEFT 90

volverá a su posición original, después de haber girado hacia la izquierda 90 grados.

Recordamos, para aquellos que no tengan mucha familiaridad con la anotación sexagesimal de los ángulos, que:

Circunferencia	=360 grados
Media circunferencia	=180 grados
Angulo recto	= 90 grados

y que:

- los ángulos internos de un triángulo equilátero (todos sus lados iguales) valen 60 grados;

- la suma de los ángulos internos de cualquier triángulo es 180 grados;
- los ángulos internos de un cuadrado o de un rectángulo valen 90 grados;
- los ángulos internos de un hexágono regular valen 120 grados.

Es correcto escribir:

RIGHT (grados de rotación a la derecha)
LEFT (grados de rotación a la izquierda)

Reflejemos a continuación un ejemplo que muestre la utilidad de los cuatro mandos básicos que acabamos de ver y que a la vez nos sirva para trazar una figura geométrica.

Para obtener un triángulo equilátero pulsemos:

LEFT 90

gira la tortuga 90 grados hacia la izquierda, disponiéndola a escribir en horizontal respecto al plano visual, pues inicialmente está "mirando" hacia arriba;

FORWARD 50

avanza 50 posiciones, trazando una línea recta horizontal que representa la mitad izquierda de la base del triángulo;

RIGHT 120

gira el cursor 120 grados a la derecha. Recordando que el ángulo llano es de 180 grados y que el ángulo interno de un triángulo equilátero es de 60, el ángulo de rotación exacto es el resultado de 180-60=120;

FORWARD 100

avanza 100 posiciones en sentido oblicuo con una inclinación respecto a la base de 60 grados, formando el lado izquierdo del triángulo;

RIGHT 120

rota 120 grados a la derecha. Para la medida de los grados vale el ejemplo anterior: 180-60=120;

FORWARD 100

avanza 100 posiciones, generando el lado derecho del triángulo;

RIGHT 120

cierra la rotación con un último ángulo de 120 grados a la derecha. La tortuga está de nuevo horizontal respecto al plano;

FORWARD 50

termina la construcción del triángulo trazando la segunda mitad del lado base (Fig. 3).

Dibujemos ahora un cuadrado.

RIGHT 90

rotando la tortuga a la derecha la ponemos sobre el plano horizontal (en el programa anterior hemos obtenido el mismo resultado con la rotación a la izquierda);

FORWARD 50

traza la mitad del lado de base;

LEFT 90

gira a la izquierda 90 grados creando el ángulo de base derecho;

FORWARD 100

avanza 100 posiciones dibujando el lado derecho;

LEFT 90

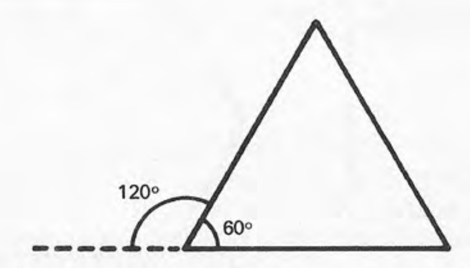


Figura 3.—Triángulo equilátero, fácilmente realizable con el LOGO.

90 grados a la izquierda. Crea el vértice superior derecho;

FORWARD 100

traza la base superior;

RIGHT 90

vamos a hacer esta operación un poco complicada; para crear el ángulo superior izquierdo sería suficiente escribir LEFT 90, pero, en cambio, de esta manera podremos mostrar la utilización de la instrucción BACK.

BACK 100

lleva la tortuga hasta la base del cuadrado, trazando el lado izquierdo;

RIGHT 90

último ángulo (inferior izquierdo). Es RIGHT, pues hemos bajado "de espaldas";

FORWARD 50

traza la segunda mitad de la base (Fig. 4).

El programa que a continuación vemos, dibuja, análogamente a los anteriores, otra figura geométrica regular (Fig. 4).

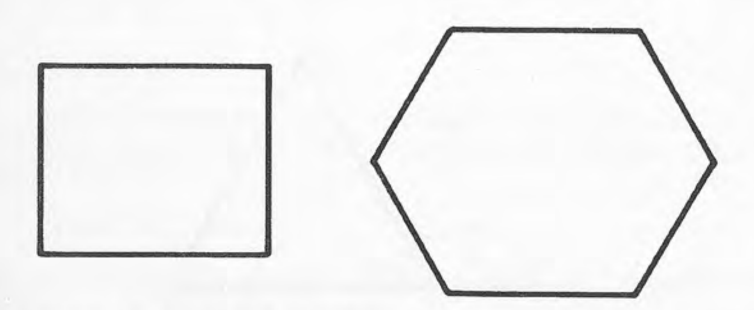


Figura 4.—Cuadrado y hexágono regular obtenidos con comandos directos.


```
LEFT 90
FORWARD 35
RIGHT 60
FORWARD 70
RIGHT 60
FORWARD 70
RIGHT 60
FORWARD 70
RIGHT 60
FORWARD 70
RIGHT 60
FORWARD 70
RIGHT 60
FORWARD 35
```

Estudiemos ahora dos instrucciones de relevante importancia para la programación gráfica, que permiten la colocación de la tortuga en *cualquier zona de la pantalla.*

Veámoslas en la forma operativa (o directa)

SETX x

lleva la tortuga hasta el punto de abscisa "x", pero no cambia la ordenada. Este ejemplo es válido, obviamente, pensando en la pantalla como en un plano sobre el cual está fijado un sistema de referencia de ejes cartesianos ortogonales, como se muestra en la figura 5.

SETY y

fuerza la ordenada de la tortuga al valor "y", *no alterando* la posición horizontal (abscisa).

SETXY x y

lleva a la tortuga hasta el punto de coordenadas cartesianas (x, y).

Es importante subrayar que si hacemos desplazarse a la tortuga desde el punto "A" hasta el punto "B" ésta dejará detrás de sí la "pista" habitual: seguidamente veremos cómo se puede evitar esto usando el comando PENUP.

Utilizando estos comandos resultará muy fácil trazar un cuadrado:

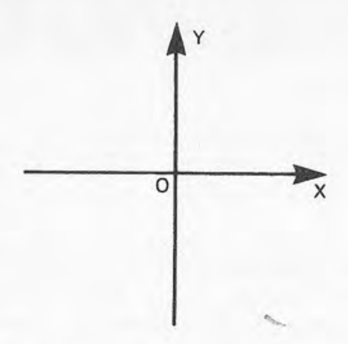


Figura 5.—Sistema cartesiano de referencia considerado para fijar la posición de un punto en la pantalla.

```
HOME
CLEARSCREEN
SETY 30
SETX 30
SETY 0
SETX 0
```

La primera instrucción, HOME, lleva la tortuga a la posición central de la pantalla (origen de los ejes) y además sitúa la cabeza de la tortuga paralelamente al eje de las ordenadas positivas (y+).

La segunda instrucción de la lista antes mencionada limpia completamente la pantalla colocada en alta resolución.

De hecho, pulsando solamente

CLEARSCREEN

desaparecerán todas las figuras anteriores del área interna gráfica sin que, de cualquier manera, resulten modificados los parámetros definidos anteriormente como color, modo pantalla y, sobre todo, posición de la tortuga.

Se ha escrito en orden antes HOME y luego CLEARSCREEN porque haciéndolo de manera inversa, después de borrar la pantalla (CLEARSCREEN) se trazaría la línea recta que une la posición de la tortuga con el centro de la pantalla (HOME), con lo cual ésta se "ensuciaría" de nuevo.

Además, existe un comando más complejo capaz de llevar a cabo simultáneamente las funciones de borrar la pantalla y de colocar de nuevo en el centro de ésta a la tortuga. Se trata (como hemos dicho ya) de:

DRAW

Vamos a analizar la siguiente serie de instrucciones, que resumen lo que se ha descrito hasta ahora dibujando una letra y que, al mismo tiempo, permite el examen de dos instrucciones nuevas.

```
HOME
CLEARSCREEN
FORWARD 120
RIGHT 90
FORWARD 40
RIGHT 90
PENUP                    [1]
FORWARD 60
RIGHT 90
PENDOWN                  [2]
FORWARD 40
LEFT 90
FORWARD 60
LEFT 90
FORWARD 40
PENDOWN                  [2]
FORWARD 40
LEFT 90
FORWARD 60
LEFT 90
FORWARD 40
```

La instrucción marcada (1), como habrán podido notar durante la ejecución del programa, ha inhibido la "mancha" de la tortuga, que se ha desplazado sin escribir ni borrar; posteriormente las marcadas (2) la han reactivado para escribir la letra E.

En resumen:

PENUP

hace que el movimiento de la tortuga no quede marcado, y

PENDOWN

es su contrario, activando de nuevo el dibujo de los movimientos.
Si al finalizar el programa anterior pulsáramos

HIDETURTLE

veríamos desaparecer de la pantalla a la tortuga. Es importante notar que aunque sea invisible sigue dejando su pista (a no ser que antes hubiéramos pulsado PENUP).
Para cerciorarnos de lo dicho escribamos:

```
HOME
CLEARSCREEN
HIDETURTLE
FORWARD 100
```

Para visualizar de nuevo la tortuga usaremos

SHOWTURTLE

Otra instrucción que actúa sobre la plumilla es

PENERASE

Como consecuencia del uso de este comando, la tortuga, al moverse, *borra* en vez de escribir.
Veamos un ejemplo de cómo utilizar correctamente esta instrucción:

```
FORWARD 100
PENERASE
BACK 100
```

Al final no queda ningún rastro: tal y como apareció, desapareció.
Pulsando seguidamente

PENCOLOR X

con X entero y positivo veremos de nuevo la pista.
Para que el cursor se vea de nuevo hay que escribir

SHOWTURTLE

Finalmente observemos el efecto de

STAMPCHAR "A

Esta instrucción permite la impresión de los caracteres estándar en alta resolución; imprime la letra en la posición siguiente a la que se encuentra la tortuga.
Por lo tanto, si escribimos

```
CLEARSCREEN
STAMPCHAR "A
```

veremos a nuestra tortuga dibujar la letra "A" en la pantalla definida en alta resolución.

Los colores

En este párrafo nos ocuparemos de instrucciones que tienen gran importancia en el contexto de una programación agradable y estéticamente válida.
Hablamos, evidentemente, de comandos que pueden cambiar las características cromáticas de la pantalla, del fondo, de las líneas trazadas en alta resolución y de la tortuga.
Examinemos, ante todo, una instrucción que cambia el color de la página de alta resolución. Escribiremos:

BACKGROUND 5 (return)

La pantalla puesta en alta resolución toma el color correspondiente al código 5.
La sintaxis, por lo tanto, es:

BACKGROUND (código-color)

Por lo que se refiere a la relación código-color, ésta varía según el ordenador y la pantalla utilizados; la tabla 1 es la de referencia para el sistema COMMODORE 64.
Si, por el contrario, deseamos cambiar los colores de la página texto utilizaremos

TEXTBG x

que en "x" especifica el color deseado (según el contenido de la tabla 1).

TEXTCOLOR x

Indica el color de los caracteres estándar.

CODIGO	COLOR
00	NEGRO
01	BLANCO
02	ROJO
03	AZUL OSCURO 1
04	PURPURA
05	VERDE 1
06	AZUL OSCURO 2
07	AMARILLO
08	NARANJA
09	MARRON
10	ROJO 1
11	GRIS 1
12	GRIS 2
13	VERDE 2
14	AZUL
15	GRIS 3

Tabla 1.—Relación código-color para el Commodore 64.

Examinemos lo que conseguimos con esta serie de instrucciones, ejecutadas de manera directa:

```
HOME
CLEARSCREEN
TEXTBG 0
```

colorea la página de texto de negro (tabla 1).

```
BACKGROUND 2
```

colorea la página de alta resolución en rojo.

```
TEXTCOLOR 13
```

da el color verde (en base a la tabla 1) a los caracteres estándar.

Archivar las imágenes

Después de haber construido una figura surge el problema de poderla conservar sobre un soporte magnético.

De esta tarea se ocupan tres comandos con funciones complementarias, que permiten todas las operaciones de "grabación", "carga" y "cancelación".

Escribamos la siguiente secuencia de instrucciones, que construyen el gráfico de ejemplo de los tres comandos antes mencionados:

```
SINGLECOLOR
FULLSCREEN
PENUP
FORWARD 100
LEFT 90
FORWARD 16 + 100
LEFT 180
STAMPCHAR "S
FORWARD 8
STAMPCHAR "A
FORWARD 8
STAMPCHAR "V
FORWARD 8
STAMPCHAR "E
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 24
LEFT 180
STAMPCHAR "R
FORWARD 8
STAMPCHAR "E
FORWARD 8
STAMPCHAR "A
FORWARD 8
STAMPCHAR "D
RIGHT 90
FORWARD 100
```

```
RIGHT 90
FORWARD 24
LEFT 180
STAMPCHAR "E
FORWARD 8
STAMPCHAR "R
FORWARD 8
STAMPCHAR "A
FORWARD 8
STAMPCHAR "S
FORWARD 8
STAMPCHAR "E
FORWARD 8
PENDOWN
LEFT 45
FORWARD 142
RIGHT 270
FORWARD 142
BACK 142
RIGHT 135
PENUP
FORWARD 8
STAMPCHAR "P
FORWARD 8
STAMPCHAR "I
FORWARD 8
STAMPCHAR "C
FORWARD 8
STAMPCHAR "T
FORWARD 16
```

En esta situación, para salvar sobre disco la página gráfica resultante (Fig. 6) escribimos:

SAVEPICT "nombre

Seguidamente, para recuperarla habría que escribir:

READPICT "nombre

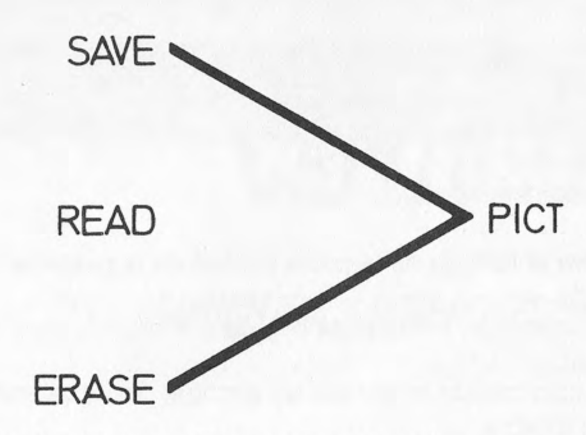



Figura 6.—Página gráfica que vamos a grabar.

En cambio, si se quiere borrar la página del disco:

ERASEPICT "nombre

Resumen de las instrucciones

DRAW
Para entrar en alta resolución.

NODRAW
Contrario de draw.

TEXTSCREEN
Vídeo completo en modo texto.

SPLITSCREEN
Vídeo subdividido en alta resolución y en modo texto.

FULLSCREEN
Vídeo completamente en alta resolución.

FORWARD x
Avanza la tortuga "x" posiciones.

BACK x
Retrasa la tortuga "x" posiciones.

RIGHT x
Gira a la derecha "x" grados la tortuga.

LEFT x
Gira a la izquierda "x" grados la tortuga.

SETX x
Sitúa la tortuga en la abscisa "x".

SETY x
Sitúa la tortuga en la ordenada "y".

CLEARSCREEN
Limpia el área gráfica.

HOME
Coloca la tortuga en la parte central de la pantalla.

PENCOLOR x
Da a la tortuga y a la pista dejada por ella el color "x".

PENERASE
Tras este mando la tortuga no escribe, sino que borra.

BACKGROUND x
Determina el color de la página gráfica.

TEXTCOLOR x
Da a los caracteres de la página texto el color deseado.

TEXTBG x
Parámetros del color para el área texto.

PENUP
Tras este comando la tortuga se desplaza sin escribir ni borrar.

PENDOWN
Lleva a cabo la función contraria a PENUP.

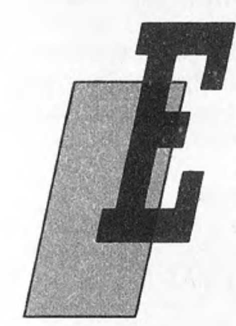
HIDETURTLE
Hace invisible la tortuga, pero no su pista.

SHOWTURTLE
Hace visible a la tortuga.

STAMPCHAR "x
Imprime en alta resolución los caracteres que siguen a las comillas.

CAPITULO II

PROCEDIMIENTOS Y SUBPROCEDIMIENTOS



En este capítulo hablaremos de una de las partes más importantes del lenguaje LOGO: los procedimientos, es decir, aquellas "palabras" que nosotros mismos podemos definir y que nos permiten usar un mismo grupo de instrucciones por medio de la sencilla mención de la palabra.

Definición de nuevas palabras

Supongamos que queremos dibujar un rectángulo como el de la figura 1.



Figura 1.—Rectángulo a dibujar.

Con los medios actualmente a nuestra disposición no podemos hacer otra cosa que pulsar secuencialmente las instrucciones:

```
FORWARD 50
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 100
```

Nos vemos obligados a escribir todo esto ya que el LOGO no pone a nuestra disposición, dentro del vocabulario de los comandos fundamentales, una instrucción que trace automáticamente la figura geométrica requerida. De hecho, este vocabulario de comandos resulta estar exclusivamente constituido por sencillas, aunque importantes, palabras llamadas PRIMITIVAS, que son las ya conocidas:

FORWARD, LEFT, RIGHT, PENCOLOR, etc.

En este contexto resalta uno de los aspectos principales del lenguaje LOGO: el programador, consecuentemente a sus propias necesidades, puede DEFINIR un conjunto de NUEVAS INSTRUCCIONES, llamadas técnicamente PROCEDIMIENTOS, que ampliarán el básico y limitado vocabulario.

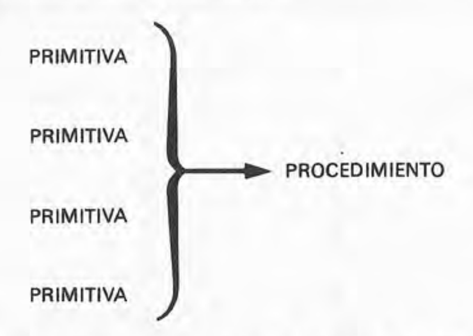


Figura 2.—El LOGO nos permite definir un procedimiento a partir de las primitivas adecuadas.

Veamos en la práctica cómo construir una nueva instrucción que, al ser llamada, dibuje en la pantalla el rectángulo de la figura 1.

Antes de todo vamos a decidir el nombre que le daremos; es aconsejable en esta fase tener presente cuatro buenas normas generales, aparentemente obvias, que en el aprendizaje resultarán de evidente utilidad y simplificarán mucho la comprensión de los programas:

1. Preferir una etiqueta que ponga bien en evidencia y si es posible SINTETICE la función del Procedimiento.
2. Buscar nombres SIMPLES y, por lo tanto, fáciles de recordar.
3. Comprobar que el nombre elegido no se confunda con otros de los Procedimientos definidos anteriormente.
4. El intérprete LOGO no permite, desde luego, que "bauticemos" un procedimiento con el apelativo típico de una primitiva o con el nombre de un Procedimiento ya presente en el vocabulario (por este motivo es imposible llamar un procedimiento "FORWARD" o "PENUP").

En el caso examinado por nosotros necesitamos un procedimiento que trace un rectángulo; el nombre que se le puede dar y que contiene los susodichos requisitos podría ser "DIBUJARECTANGULO".

Lo hacemos por medio de la instrucción:

TO DIBUJARECTANGULO

En este momento podemos introducir secuencialmente las instrucciones que deseamos que se memoricen en relación a nuestro Procedimiento.

Ahora, encontrándonos ya en el ambiente EDITOR, los comandos pulsados no se llevarán a cabo inmediatamente. Sólo se memorizarán y se asignarán a la palabra que estamos definiendo; será después, cuando llamemos a nuestro Procedimiento, cuando esta secuencia de mandos se llevará a cabo automáticamente y de forma sucesiva sin tener que escribirla de nuevo por entero.

Después de haber escrito el TO DIBUJARECTANGULO y aclarado estos puntos, pulsemos:

```
RIGHT 90 ;[REALIZA UN GIRO DE 90 GRADOS]
FORWARD 90 ;[AVANZA 90 POSICIONES]
LEFT 90 ;[GIRO A LA IZQUIERDA]
FORWARD 40 ;[AVANZA EN ESA DIRECCION]
LEFT 90 ;[OTROS 90 GRADOS]
FORWARD 90 ;[LADO SUPERIOR]
```

```
LEFT 90 ;[ULTIMO GIRO]
FORWARD 40 ;[ULTIMO LADO]
END
```

En el ejemplo podemos observar dos detalles importantes.

- En la definición de un procedimiento cualquiera se permite realizar comentarios de cualquier tipo, a condición de que estén cerrados entre paréntesis cuadrados y precedidos por el signo ";"
- La última instrucción de un procedimiento debe ser necesariamente END; en el caso de que se omita, será el EDITOR quien se encargará de añadirla al final del listado.

Pulsemos ahora, a la vez, las teclas

<CTRL> y <C>

para que el Procedimiento en cuestión se memorice y, por lo tanto, entre en el vocabulario LOGO.

En este punto la computadora contesta:

PLEASE WAIT...

y en seguida, después, si todo ha sido introducido correctamente:

DIBUJARECTANGULO DEFINED

Hemos vuelto al *modo indirecto*, o sea, a un ambiente de programación en el cual los mandos pulsados no se memorizan, sino que son llevados a cabo *inmediatamente*, al contrario de lo que pasaba en el EDITOR.

Llamemos ahora el Procedimiento recién construido con

RUN DIBUJARECTANGULO

que reproduce en la pantalla el rectángulo de la figura 3.

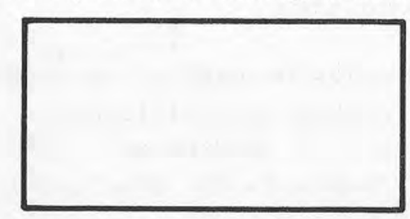


Figura 3.—Rectángulo obtenido con el procedimiento DIBUJARECTANGULO.

Ahora que hemos entendido la utilidad y las operaciones necesarias en relación a la creación de un nuevo mando podemos, por analogía, discutir los subprocedimientos, es decir, los procedimientos que participan en la definición ulterior de nuevas instrucciones.

Si, por ejemplo, queremos dibujar en la pantalla cuatro rectángulos colocados como en la figura 4, podremos definir un procedimiento que llame cuatro veces el procedimiento "DIBUJARECTANGULO" definido anteriormente y que, en este caso, toma la función de subprocedimiento. El diagrama de sintaxis es el mostrado en la figura 5.

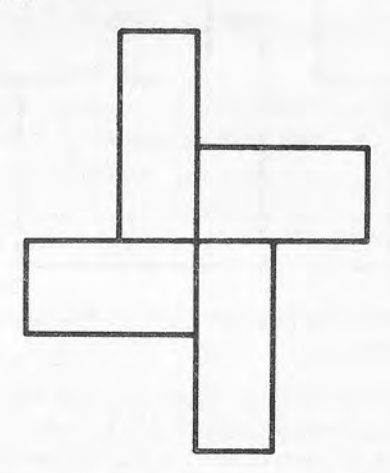


Figura 4.—Un procedimiento puede usar de otros, a los que convierte en subprocedimientos, para sus fines.

Por lo tanto podemos escribir

```
TO CUATRO.RECTANGULOS
  DIBUJARECTANGULO ;[PRIMERO]
  DIBUJARECTANGULO ;[SEGUNDO]
  DIBUJARECTANGULO ;[TERCERO]
  DIBUJARECTANGULO ;[CUARTO]
END
```

y pulsando simultáneamente

<CTRL> y <C>

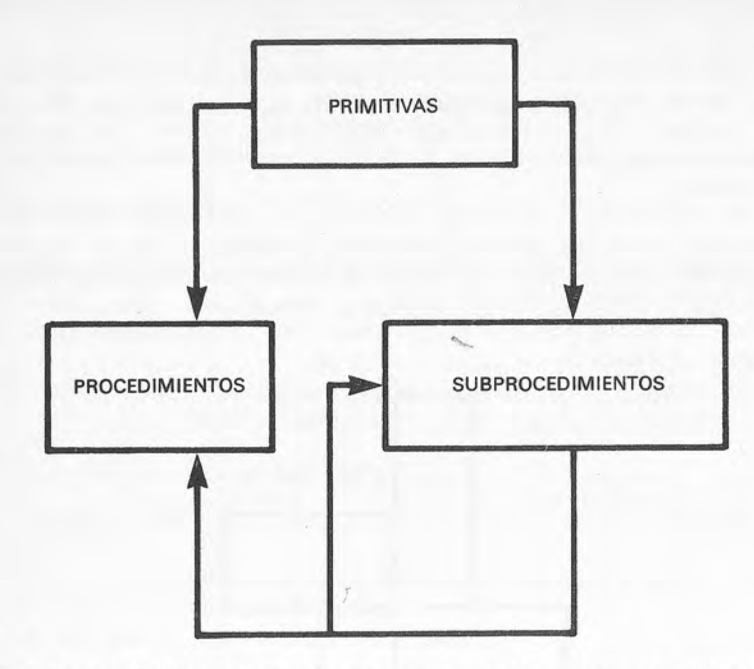


Figura 5.—Gráfico de relación entre primitivas, subprocedimientos y procedimientos.

memorizamos la nueva palabra-llave llamada "CUATRO.RECTANGULOS". El procedimiento CUATRO.RECTANGULOS no explota completamente las capacidades del LOGO; en los capítulos que siguen veremos cómo existe una instrucción que nos puede evitar la repetición de instrucciones idénticas.

Así, para conseguir lo que tenemos en la figura 4 será suficiente escribir:

RUN CUATRO.RECTANGULOS

Está claro que pudiendo definir a nuestro propio gusto instrucciones, el lenguaje LOGO permite una sensible flexibilidad en relación a cada problema de programación.

He aquí unos ejemplos, que pueden constituir para el lector interesado una válida ayuda, resumiendo lo aprendido hasta ahora.

```
TO LOGO

  HOME ;[LLEVA LA TORTUGA AL CENTRO DE
     LA PANTALLA]
```



```
CLEARSCREEN ;[BORRA POR COMPLETO TODA
   LA PANTALLA]
;[COMIENZA EL PROGRAMA DE DISENO DE LA
   PALABRA LOGO]
PENUP ;[ALZA EL LAPIZ PARA QUE EL
   RECORRIDO DE LA TORTUGA NO SEA
   VISUALIZADO]
LEFT 90 ;[GIRO DE 90 GRADOS A LA
   IZQUIERDA]
FORWARD 70 ;[AVANZA HACIA ADELANTE 70
   POSICIONES]
PENDOWN ;[UNA VEZ SITUADA LA TORTUGA
   DONDE QUERIAMOS BAJAMOS EL LAPIZ
   PARA DIBUJAR SU RASTRO]
FORWARD 40 ;[DIBUJAMOS LA 'L']
RIGHT 90 ;[GIRO A LA DERECHA DE 90]
FORWARD 70
RIGHT 90
FORWARD 10
RIGHT 90
FORWARD 60
LEFT 90
FORWARD 30
RIGHT 90
FORWARD 10
PENUP ;[ACABADA LA 'L' PREPARAMOS
   TODO PARA LA 'O', LEVANTANDO EL
   LAPIZ ANTES]
LEFT 90
FORWARD 20
LEFT 90
O ;[LLAMAMOS AL SUBPROCEDIMIENTO 'O']
PENDOWN
FORWARD 40 ;[DIBUJAMOS LA 'G']
LEFT 90
FORWARD 20
```

```
  LEFT 90
  FORWARD 10
  LEFT 90
  FORWARD 10
  RIGHT 90
  FORWARD 20
  RIGHT 90
  FORWARD 50
  RIGHT 90
  FORWARD 30
  LEFT 90
  FORWARD 10
  LEFT 90
  FORWARD 40
  LEFT 90
  FORWARD 70
  LEFT 90
  PENUP   ;[LO PREPARAMOS PARA LA
     SEGUNDA 'O']
  FORWARD 60
  LEFT 90
  O ;[NUEVA LLAMADA AL SUBPROCEDIMIENTO
     'O']
END

TO O

  PENDOWN ;[EL PROCEDIMIENTO DIBUJA LA
     LETRA 'O']
  FORWARD 70
  RIGHT 90
  FORWARD 40
  RIGHT 90
  FORWARD 70
  RIGHT 90
  FORWARD 40
```



```
  PENUP ;[LEVANTAMOS EL LAPIZ PARA
     SITUARNOS EN LA PARTE INETRIOR]
  RIGHT 90
  FORWARD 10
  RIGHT 90
  FORWARD 10
  PENDOWN
  FORWARD 20
  LEFT 90
  FORWARD 50
  LEFT 90
  FORWARD 20
  LEFT 90
  FORWARD 50
  PENUP ;[POSICIONA LA TORTUGA EN EL
     COMIENZO DE LA SIGUIENTE LETRA]
  FORWARD 10
  LEFT 90
  FORWARD 50

END
```

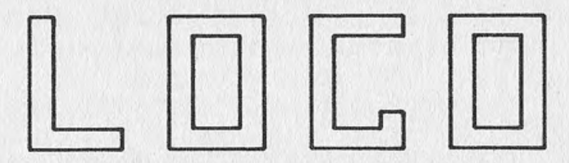


Figura 6.—Resultado de la ejecución de un procedimiento que, a su vez, hace uso de subprocedimientos.

El editor

Con este término técnico se indica aquella sección del lenguaje cuya función específica es la de permitir y ayudar al máximo al usuario cuando éste escribe procedimientos, subprocedimientos y, por lo tanto, cualquier programa.

En el momento de encender el ordenador se sitúa en modo directo o de comandos (COMMAND MODE); en él, si nosotros pul-

samos una instrucción correcta, ésta se llevará a cabo inmediatamente en cuanto pulsemos la tecla RETURN.

Por ejemplo, escribiendo:

FORWARD 50 (y el necesario <RETURN>)

veremos, después de unos momentos, cómo la tortuga se mueve en la pantalla 50 posiciones.

Sin embargo, si pulsamos

TO NOMBREPROCEDIMIENTO

o bien

EDIT NOMBREPROCEDIMIENTO

o también

EDIT ALL

se entra en modo editor, es decir, podemos introducir y/o modificar las secuencias de instrucciones relativas al procedimiento que deseemos.

Cuando más tarde llamemos a este procedimiento, las instrucciones dadas serán llevadas a cabo en secuencia fielmente, como si nosotros las estuviéramos pulsando en el modo directo.

Examinemos ahora cuáles son las opciones a disposición del programador LOGO en el caso de que haya entrado en ambiente EDITOR.

A la izquierda les ofrecemos una lista de las teclas a pulsar simultáneamente, y a la derecha se explica su función (para el CBM 64).

CTRL y O	Equivale a pulsar RETURN; se introduce una línea vacía en el texto del Programa y se llevan a nueva línea todos los caracteres sucesivos al cursor.
CTRL y N	Pulsando simultáneamente estas dos teclas, el cursor se sitúa al inicio de la línea SUCESIVA con respecto a la que se encuentra.
CTRL y P	Efectúa la función contraria a la del comando anterior, moviendo el cursor al inicio de la línea ANTERIOR.
CTRL y L	Coloca el cursor al final de la línea actual.
CTRL y A	Desplaza el cursor al extremo izquierdo de la pantalla.

CTRL y F	Lleva el cursor al final del programa.
CTRL y B	Mueve el cursor hasta el primer carácter del programa.
CTRL y D	Se borra el carácter sobre el que está colocado el cursor.
CTRL y K	Borra todos los caracteres a la DERECHA del cursor.

Finalmente, muy importantes, son las instrucciones:

CTRL y C
CTRL y G

la primera define el procedimiento que se está llevando a cabo y las eventuales variaciones, mientras que la segunda da por acabada la fase de variación y/o modificación. Estos dos últimos comandos TRANSFIEREN DE NUEVO el control a la manera INMEDIATA. La figura 7 ilustra este concepto.

Listado y borrado de procedimientos

En los párrafos anteriores hemos hablado detalladamente de las operaciones necesarias para la definición y la modificación de un procedimiento y/o subprocedimiento. Sin embargo, muy a menudo pasa que, después de haber escrito un programa y haber comprobado sus resultados, nace la necesidad de visualizar velozmente el listado correspondiente.

Para hacer todo esto es fundamental poder tener a nuestra disposición el nombre de todos los procedimientos presentes hasta el momento en el vocabulario. La instrucción a la cual se confió este cometido es:

POTS

Si, por ejemplo, nuestro vocabulario está constituido por los procedimientos "DIBUJARECTANGULO", "CUATRO.RECTANGULOS" y "TRIANGULO", pulsando POTS obtendremos en la pantalla:

```
?POTS
TO DIBUJARECTANGULO
TO CUATRO.RECTANGULOS
TO TRIANGULO
```

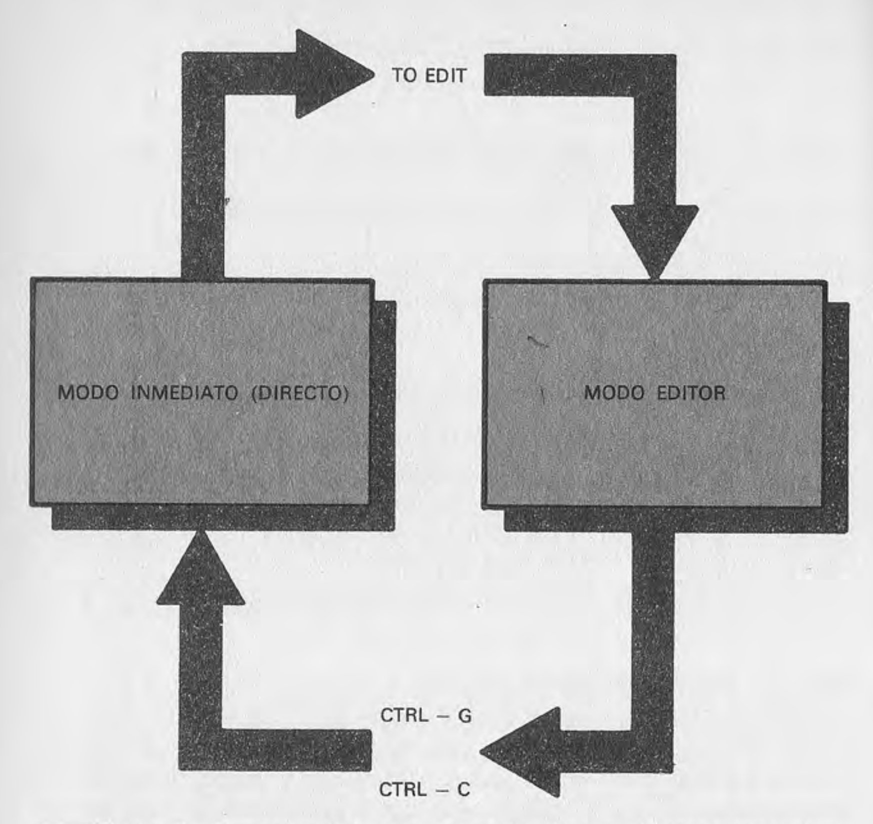


Figura 7.—Esquema del paso de modo directo a editor y viceversa. Los comandos son los del CBM64.

Para conseguir el listado completo de un procedimiento ya escrito se usa el comando PO, cuya sintaxis es:

PO NOMBREPROCEDIMIENTO

Esta operación puede aparecer a simple vista igual que la instrucción TO NOMBREPROCEDIMIENTO, pero, en realidad, es muy distinta.

De hecho, mientras TO permite la más o menos trabajosa modificación del listado, ya que pasa el control de las operaciones al EDITOR, PO permite exclusivamente la visualización de ese listado.

Si pulsamos ahora:

PO ALL

aparecerá en la pantalla el listado de todos los procedimientos residentes en memoria, que aparecen en el orden contrario con respecto al que se definieron. En otras palabras: la última palabra creada por el programador será la primera en ser listada.

Es útil recordar que el "scroll" (desplazamiento) de la pantalla puede ser momentáneamente interrumpido pulsando a la vez

CTRL y W

y activado de nuevo con sólo pulsar una tecla cualquiera.

Ahora que hemos examinado cómo INTRODUCIR, LISTAR y MODIFICAR un procedimiento, nos queda tomar en consideración la fase de anulación, parcial o total, del VOCABULARIO.

Este cometido, desde luego muy importante (recordemos que anular un procedimiento significa ganar memoria), está confiado a la instrucción ERASE, cuya sintaxis es:

ERASE NOMBREPROCEDIMIENTO

Con ella eliminamos (y desaparece así del vocabulario) la palabra de la cual hemos especificado el nombre, a condición que no sea una primitiva.

ERASE ALL

borra TODOS los procedimientos residentes en memoria, llevando el vocabulario LOGO a las condiciones típicas que siguen a la puesta en marcha de nuestro ordenador.

Grabar el vocabulario

Cuando se han definido palabras nuevas de uso frecuente o que volveremos a usar es necesario poderlas almacenar para luego tenerlas a nuestra disposición en cualquier momento sin estar obligado a volverlas a escribir y verificar.

El LOGO nos permite la memorización de los procedimientos en soporte magnético (de manera particular en disquetes —Floppy Disk— y en cinta) para crear un archivo personal de nuevas instrucciones.

Si queremos "salvar" un vocabulario (FICHERO) creado por nosotros deberemos usar el comando:

SAVE "NOMBREFICHERO

Para averiguar si la grabación del fichero-vocabulario ha sido lle-

vada a cabo correctamente llamamos al directorio (directory) del disco pulsando la instrucción:

CATALOG

Veremos cómo aparece el nombre de nuestro fichero.

Carguemos ahora en memoria el vocabulario salvado.

Para estar seguros de que esta operación se realiza y no ser engañados por el hecho de que el vocabulario está ya en memoria, llevamos el ordenador a las condiciones iniciales (de encendido), de forma que esté sin ningún procedimiento predefinido:

GOODBYE

La máquina se inicializará. Llegados aquí utilizamos la instrucción

READ "NOMBRE

para que reaparezca el fichero-vocabulario anteriormente archivado.

Para hacer más comprensible lo dicho hasta ahora hagamos un ejemplo práctico; consideremos un vocabulario constituido por los procedimientos DIBUJARECTANGULO y CUATRORECTANGULOS.

Para grabarlo pulsaremos

SAVE "PRIMER-VOCABULARIO"

Para cargarlo:

READ "PRIMER-VOCABULARIO"

Es importante tener en cuenta dos cosas:

1. Es posible salvar el vocabulario con un nombre distinto de los procedimientos en él contenidos.
2. En el caso en que se cargue en memoria un segundo vocabulario, éste se junta al anterior sin modificarlo. Ahora bien, ¡cuidado con no cargar dos procedimientos que tengan el mismo nombre!, pues en ese caso se produciría la cancelación del último.

Queda por discutir el borrado de las palabras salvadas sobre disco. Esta operación, llamada SCRATCH, está confiada al comando:

ERASEFICHERO "NOMBRE

Programas resumen

Programa 1 - Demostración

El siguiente programa traza el gráfico de las operaciones realizadas entre la memoria y el fichero en disco o casete siguientes:

Definición procedimientos
Salvar fichero
Cargar ficheros en memoria

Les recordamos que después de cada línea existe un <RETURN>.

```
TO D
  ;[ESTE PROCEDIMIENTO ESCRIBE LA LETRA
     'D' Y MUEVE LA TORTUGA 8
     POSICIONES]
  STAMPCHAR "D
  FORWARD 8
END

TO L
  ;[ESCRIBE LA 'L']
  STAMPCHAR "L
  FORWARD 8
END

TO F
  ;[ESCRIBE LA 'F']
  STAMPCHAR "F
  FORWARD 8
END

TO V
  ;[ESCRIBE LA 'V']
  STAMPCHAR "V
  FORWARD 8
END
```

```
TO S
  ;[ESCRIBE LA 'S']
  STAMPCHAR "S
  FORWARD 8
END

TO A
  ;[ESCRIBE LA 'A']
  STAMPCHAR "A
  FORWARD 8
END

TO I
  ;[ESCRIBE LA 'I']
  STAMPCHAR "I
  FORWARD 8
END

TO R
  ;[ESCRIBE LA 'R']
  STAMPCHAR "R
  FORWARD 8
END

TO O
  ;[ESCRIBE LA 'O']
  STAMPCHAR "O
  FORWARD 8
END

TO E
  ;[ESCRIBE LA 'E']
  STAMPCHAR "E
  FORWARD 8
END
```



```
TO M
  ;[ESCRIBE LA 'M']
  STAMPCHAR "M
  FORWARD 8
END

TO READ

  ;[ESCRIBE LA PALABRA 'READ']
  RIGHT 90
  FORWARD 70
  LEFT 90
  BACK 130
  R ;[LLAMA AL SUBPROCEDIMIENTO 'R']
  E
  A
  D
  FORWARD 20

END

TO FILE

  ;[ESCRIBE LA PALABRA 'FILE']
  FORWARD 65
  RIGHT 90
  FORWARD 70
  LEFT 90
  F
  I
  L
  E

END
```

```
TO SAVE

   ;[ESCRIBE LA PALABRA 'SAVE']
   LEFT 90
   FORWARD 70
   RIGHT 90
   FORWARD 40
   S
   A
   V
   E

END

TO MEMORIA

   ;[ESCRIBE LA PALABRA 'MEMORIA']
   FORWARD 30
   RIGHT 90
   FORWARD 205
   LEFT 180
   M
   E
   M
   O
   R
   I
   A

END

TO RECTANGULO

   ;[DIBUJA UN RECTANGULO]
   RIGHT 90
   FORWARD 60
   LEFT 90
   FORWARD 60
```



```
   LEFT 90
   FORWARD 120
   LEFT 90
   FORWARD 60
   LEFT 90
   FORWARD 60

END

TO DEMOSTRACION

   ;[LLAMA Y CONTROLA TODOS LOS
     SUBPROCEDIMIENTOS ANTERIORES]
   HOME ;[TORTUGA AL CENTRO DE LA
     PANTALLA ]
   CLEARSCREEN ;[BORRA LA PANTALLA]
   PENUP ;[LEVANTA EL LAPIZ PARA NO
     MARCAR EL RECORRIDO]
   RIGHT 90
   FORWARD 80
   PENDOWN ;[SITUADA LA TORTUGA VUELVE A
     MARCAR SU RASTRO]
   LEFT 90
   RECTANGULO ;[LLAMA AL SUBPROCEDIMIENTO
     'RECTANGULO']
   RIGHT 90
   FORWARD 30
   RIGHT 90
   FORWARD 180
   RIGHT 90
   FORWARD 30
   RECTANGULO
   PENUP
   LEFT 90
   FORWARD 60
   PENDOWN
```

```
FORWARD 30
RIGHT 90
FORWARD 180
RIGHT 90
FORWARD 30
PENUP
MEMORIA ;[LLAMADA AL SUBPROCEDIMIENTO
  'MEMORIA']
SAVE ;[LLAMADA AL SUBPROCEDIMIENTO
  'SAVE']
FILE ;[LLAMADA AL SUBPROCEDIMIENTO
  'FILE']
READ ;[LLAMADA AL SUBPROCEDIMIENTO
  'READ']

END
```

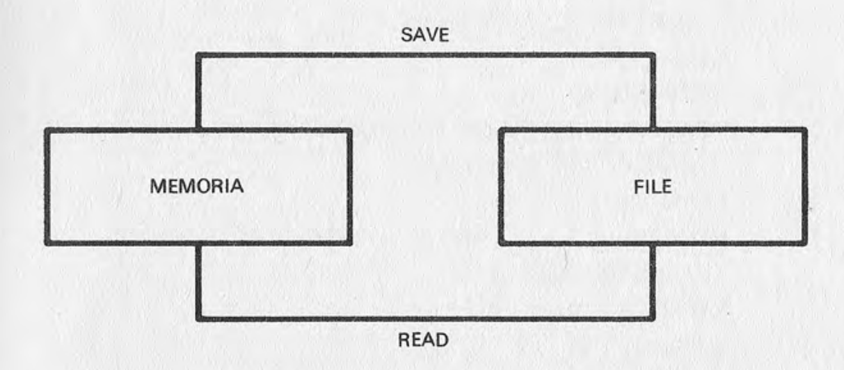


Figura 8.—Resultado del programa DEMOSTRACION.

Programa 2 - Hoja

El programa HOJA dibuja, como dice su nombre, una hoja.

En un primer momento se crea un procedimiento que dibuja un rombo y éste es llamado después repetidas veces, aunque con una única instrucción (REPEAT) que se verá posteriormente.


```
TO HOJA

  HOME ;[TORTUGA AL CENTRO]
  CLEARSCREEN ;[BORRA PANTALLA]
  PENUP ;[LEVANTA LA PLUMILLA]
  FORWARD 10
  PENDOWN ;[BAJA LA PLUMILLA]
  ROTACION ;[LLAMADA AL SUBPROCEDIMIENTO
    'ROTACION']

END


TO ROTACION

  REPEAT 7 [ROMBO] ;[ESPERE A VER EL
    CAPITULO 5]
  RIGHT 140
  FORWARD 50
  LEFT 40
  FORWARD 50
  RIGHT 220
  FORWARD 50
  LEFT 40
  FORWARD 50

END


TO ROMBO

  LEFT 150
  FORWARD 60
  LEFT 60
  FORWARD 60
  RIGHT 240
```

```
    FORWARD 60
    LEFT 60
    FORWARD 60

END
```

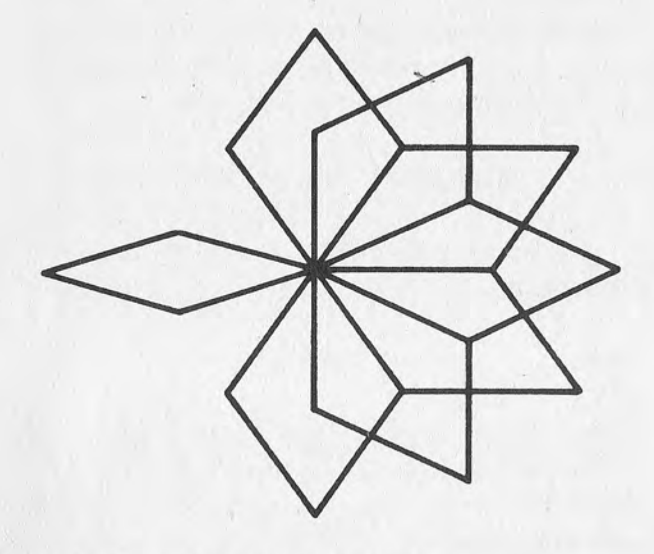


Figura 9.—Ejecución del programa HOJA.

CAPITULO III

CONSTANTES, VARIABLES Y TABLAS

"Make" y "Print"

Como todo lenguaje evolucionado, el LOGO ofrece la posibilidad de tratar a nivel de programa constantes y variables con relativa simplicidad. Examinemos en detalle las palabras del lenguaje que desarrollan este aspecto operativo.

Hay que tener presente que al definir una constante o variable no podemos asignarle ninguno de los nombres característicos del LOGO (o sea, primitivas y procedimientos).

Además, es necesario recordar que el valor numérico atribuible a constantes o variables debe de estar necesariamente comprendido entre un mínimo y un máximo, típicos de la máquina con la cual se trabaja.

Una variable que en el transcurso del programa asume un valor no consentido genera la inmediata detención de la ejecución del procedimiento en curso y la impresión del mensaje

NUMBER TOO BIG (o TOO SMALL)

Consideremos ahora en la práctica el procedimiento necesario para la definición de una constante, que en el ejemplo es llamada "CONS" y que debe asumir el valor numérico entero 150.

Es suficiente pulsar

MAKE "CONST 150

La instrucción MAKE asigna a la palabra siguiente el valor especificado; éste no tiene que ser necesariamente una constante,

ya que puede ser cómodamente sustituido con una expresión, como en el ejemplo siguiente:

MAKE "ALFA (2+3*5)

O también:

MAKE "BETA SIN (180) * COS (0)

Otro aspecto fundamental de la instrucción MAKE es que permite también la creación de constantes y variables con argumento alfanumérico.

La sintaxis a seguir en este caso es ligeramente diferente y se la puede sintetizar así:

MAKE "NOMBRE "ARGUMENTO

Queriendo asignar la frase "21.Enero.1986" a la constante "FECHA", se escribe:

MAKE "FECHA "21.ENERO 1986

El programador menos experto debe de tener cuidado al utilizar la instrucción MAKE con una secuencia de instrucciones del tipo

MAKE "ALFA 10
MAKE "ALFA 20

que llevaría, en primer lugar, a la asignación del valor numérico 10 a la constante ALFA e, inmediatamente después, a la colocación del valor 20 en la misma constante, con la obvia pérdida completa de la información anterior (10).

Es éste un ejemplo que, si bien inicialmente puede parecer simple y sin sentido, puede tal vez hacer perder tiempo en la corrección de un programa incluso al usuario más experto.

Paralelamente a la instrucción MAKE examinemos la primitiva del lenguaje cuya función es prácticamente la opuesta, o sea, la de imprimir el valor del argumento y no la de asignar un valor a una variable o constante.

La instrucción en cuestión es del tipo

PRINT VALOR

o también

PRINT :NOMBREVARIABLE

Puede ser utilizada de modo directo o como programa. Probemos, por lo tanto, la pulsación de

MAKE "FRASE "QUERIDO AMIGO

y

PRINT :FRASE

se imprime, evidentemente: Querido amigo.

Para imprimir mensajes directamente se usa la sintaxis

PRINT [FRASE A IMPRIMIR]

El usuario más experto se habrá dado cuenta seguramente del hecho de que si nosotros utilizamos repetidamente el comando PRINT, los distintos argumentos se imprimen sobre diferentes líneas de la pantalla.

Pulse, por ejemplo, el procedimiento:

```
TO IMPRIMEFRASE
 PRINT [ESTOY APRENDIENDO]
 PRINT [A UTILIZAR]
 PRINT [EL LOGO]
END
```

Después de haberlo ejecutado, en la pantalla aparece un mensaje en tres líneas del tipo:

```
ESTOY APRENDIENDO
A UTILIZAR
EL LOGO
```

Si deseamos que nuestro mensaje se disponga sobre una sola línea, recurriremos a la primitiva PRINT1, del todo similar, por lo que concierne a la sintaxis, a la PRINT examinada anteriormente.

Volvamos a escribir, por lo tanto, nuestro procedimiento como abajo se indica:

```
TO IMPRIMEFRASE
 PRINT1 [ESTOY APRENDIENDO]
```

```
 PRINT1 [A UTILIZAR]
 PRINT1 [EL LOGO]
END
```

La impresión sobre pantalla es, como nos esperamos:

ESTOY APRENDIENDO A UTILIZAR EL LOGO

Variables locales: LOCAL

Concluido el tema sobre MAKE y PRINT, las fundamentales instrucciones de manipulación de las variables numéricas y alfanuméricas, bajemos a detalles técnicos y examinemos la utilización de una nueva primitiva (LOCAL) que reviste también una tarea de particular interés.

Muy frecuentemente ocurre que un subprocedimiento trabaja sobre una o más variables utilizadas por el programa principal y cuyo valor no debe de ser alterado; es, por lo tanto, indispensable que al final de la ejecución del procedimiento la variable en cuestión reasuma el valor que tenía en el momento del requerimiento.

El lenguaje LOGO resuelve con extrema simplicidad este problema a través de la instrucción:

LOCAL "NOMBREVARIABLE

Para entender a fondo la función de esta primitiva examinemos el siguiente programa:

```
TO SUBPROCEDIMIENTO
 LOCAL "ALFA
 MAKE "ALFA 6
 PRINT :ALFA
END

TO PROGRAMAPRINCIPAL
 MAKE "ALFA 5
```



```
 SUBPROCEDIMIENTO
 PRINT :ALFA
END
```

Le vamos a ejecutar con el acostumbrado

RUN PROGRAMAPRINCIPAL

Los pasos que se irán desarrollando serán entonces:

1. MAKE da el valor 5 a ALFA.
2. Se llama al subprocedimiento llamado SUBPROCEDIMIENTO.
3. La instrucción LOCAL indica que nuestro subprocedimiento trabajará con la variable ALFA utilizada por el programa principal y que, por lo tanto, al final de la ejecución del subprocedimiento deberá retomar automáticamente el valor original (que es 5).
4. El subprocedimiento está en acción: alfa ahora vale 6 (MAKE "ALFA 6).
5. Se imprime el valor correspondiente (6).
6. Se vuelve al programa principal: el valor de ALFA se pone automáticamente en 5.
7. Una vez impreso ALFA el programa termina.

"REQUEST" y "READCHARACTER"

Al introducir en un programa la instrucción "REQUEST" se para momentáneamente su ejecución hasta que el operador pulse la tecla "RETURN".

La función de esta primitiva es pedir al usuario una información que luego será utilizada por el programa; si queremos definir un procedimiento que pregunte un número y que imprima su cuadrado, pulsaremos

```
TO CUADRADO
 MAKE "X REQUEST ;[X = TECLAS ANTES DEL RETURN]
 PRINT1 [EL CUADRADO ES: ]
PRINT :X * :X
END
```

Activamos el procedimiento con RUN CUADRADO, pulsamos un ocho y "RETURN"; la contestación será:

EL CUADRADO ES 64

Al contrario de la primitiva "REQUEST", la instrucción "READCHARACTER" espera un solo dato de entrada; el programa se para hasta que no pulsemos una tecla cualquiera.

El ejemplo puede sugerir al lector una de las distintas maneras de utilización de esta primitiva. El breve programa, llamado COMANDOS.DIRECTOS, presentado más abajo, permite dibujar en la pantalla presionando las teclas "A"=adelante, "B"=back, atrás, "D"=gira a la derecha, "I" gira a la izquierda, "F"=final de procedimiento.

Por lo que se refiere al uso de la instrucción IF..THEN (utilizada en el programa) se aconseja al lector esperar a leer el capítulo 4.

```
TO COMANDOS.DIRECTOS
 HOME ;[TORTUGA A LA POSICION INICIAL]
 CLEARSCREEN ;[BORRAMOS LA PANTALLA]
 PRINT [PULSAR A, B, D, I ( F = FINAL ) ]
 GETCARACTER ;[SUBPROCEDIMIENTO PARA
 REALIZAR LA LECTURA DE LOS COMANDOS]
END

TO GETCARACTER
 MAKE "TECLAPULSADA READCHARACTER
 IF :TECLAPULSADA = "F THEN NODRAW
   PRINT[] PRINT [] PRINT [FIN] STOP
 IF :TECLAPULSADA = "A THEN PRINT
   [ADELANTE] FORWARD 10
 IF :TECLAPULSADA = "B THEN PRINT [ATRAS]
   BACK 10
```



```
 IF :TECLAPULSADA = "D THEN PRINT [DERECHA]
   RIGHT 15
 IF :TECLAPULSADA = "I THEN PRINT
   [IZQUIERDA] LEFT 15
 GET
END
```

Procedimientos con variables

Ya que hemos entendido cómo se pueden utilizar las variables, examinamos ahora cuáles son las técnicas que permiten utilizarlas en un programa completo.

Hasta ahora hemos tomado siempre en consideración la utilización de primitivas gráficas con constantes, como: FORWARD 10, LEFT 50, PENCOLOR 1, etc.

El LOGO permite asignar a una primitiva valores VARIABLES a condición de que el nombre de la variable en cuestión esté precedido por ":".

Haciendo referencia al ejemplo mencionado antes obtendremos, por tanto:

FORWARD :X, LEFT :Y, PENCOLOR :Z...

donde las variables "X", "Y", "Z" valen, respectivamente (MAKE) 10, 50 y 1.

Están permitidas, además, secuencias de instrucciones del tipo:

MAKE "ALFA 10 MAKE "BETA 40
PRINT :ALFA - :BETA

Tenga muy en cuenta que si en una expresión figura el nombre de una variable, hay que poner necesariamente delante de ella el símbolo :

Se intuye fácilmente la gran eficacia operativa de un funcionamiento tal.

Por ejemplo, si queremos dibujar N figuras similares, pero de diferentes dimensiones, sólo debemos definir un único procedimiento, capaz de trabajar sobre PARAMETROS VARIABLES (ALTURA, LADO); después será suficiente volverlo a llamar N veces

pasándole los valores dimensionales apropiados para conseguir los N dibujos deseados.

Por ejemplo, definamos un procedimiento que nos permita dibujar fácilmente un rectángulo con las dimensiones deseadas:

```
TO RECTANGULO :BASE :ALTURA
 HOME ;[CENTRA LA TORTUGA]
 CLEARSCREEN ;[LIMPIA LA PANTALLA]
 FORWARD :ALTURA ;[LADO IZQUIERDO]
 RIGHT 90 ;[GIRA A LA DERECHA]
 FORWARD :BASE ;[BASE SUPERIOR]
 RIGHT 90 ;[NUEVO GIRO]
 FORWARD :ALTURA ;[LADO DERECHO]
 RIGHT 90
 FORWARD :BASE ;[BASE]
 RIGHT 90 ;[TORTUGA EN LA POSICION INICIAL]
END
```

Ahora lo llamamos dándole como entradas (INPUT) los datos necesarios para llevar a cabo su cometido.

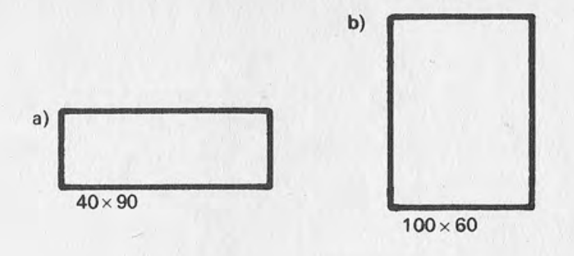
Por ejemplo, escribamos: RUN RECTANGULO 40 90 (fig. 1a).
RUN RECTANGULO 90 40

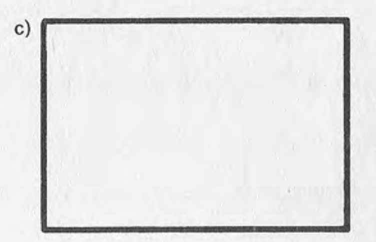
o finalmente, para conseguir un cuadrado mediante un rectángulo con altura y base iguales, RUN RECTANGULO 90 90 (Fig. 1d).

Tomemos en consideración ahora la utilización de las variables en un subprocedimiento.

El programa presentado como ejemplo (TRES.RECTANGULOS) llama tres veces el subprocedimiento RECTANGULO anteriormente definido, proporcionándole las dimensiones de la base y de la altura.

```
TO TRES.RECTANGULOS
 RECTANGULO :BASE :ALTURA
 RIGHT 120
```



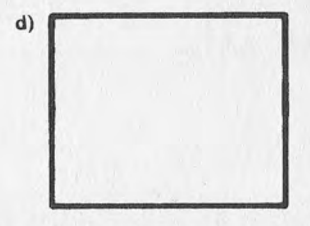


Figura 1.—Distintos resultados de ejecutar el procedimiento RECTANGULO con diversos valores.

```
 RECTANGULO :BASE :ALTURA
 RIGHT 120
 RECTANGULO :BASE :ALTURA
 RIGHT 120
END
```

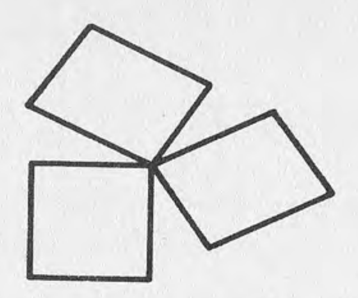


 *Figura 2.—Resultado del procedimiento TRES.RECTANGULOS.*

La imagen dibujada por el susodicho programa está reproducida en la figura 2.

La instrucción "OUTPUT"

La función del comando "OUTPUT" se desarrolla únicamente en modo programación (no es lícito utilizarlo de manera directa); devuelve un valor desde el procedimiento que lo ha utilizado.

La sintaxis de esta instrucción es la siguiente:

OUTPUT CONSTANTE

O también

OUTPUT :VARIABLE

Si llevamos a cabo la ejecución del procedimiento:

```
TO EJEMPLO
 OUTPUT 10
END
```

con RUN EJEMPLO, en la pantalla aparecerá el mensaje:

RESULT : 10

Esto es porque la instrucción "OUTPUT" fuerza el valor especificado (10) desde el procedimiento que le ha utilizado.

Una manera de utilizar la instrucción OUTPUT podría ser la siguiente:

```
TO CALCULA.TANGENTE.DE :ARGUMENTO
 OUTPUT (SIN (:ARGUMENTO)/COS (:ARGUMENTO))
END
```

Llevemos a cabo el procedimiento con:

RUN CALCULA.TANGENTE.DE 130

y la computadora contestará con

RESULT : -1.1918

En los ejemplos anteriores hemos visto cómo el LOGO imprime en la pantalla el valor derivado de la llamada a un procedimiento que utiliza la instrucción OUTPUT.

Es posible realizar el proceso de manera que el valor generado por OUTPUT no sea impreso pero sí empleado como argumento de ulteriores primitivas o procedimientos, como resulta en el ejemplo que sigue. Pulsando de manera directa

FORWARD 50 * CALCULA.TANGENTE.DE 140

la tortuga se desplazará hacia delante un número de posiciones igual a

12=50 * (tangente de 140)

En el capítulo 4 veremos de cuánta potencia operativa resulta ser la instrucción OUTPUT si se introduce en programas que utilizan técnicas recurrentes.

Tratamiento de las listas y tablas

Una considerable parte del lenguaje LOGO está orientada a permitir al programador un dominio eficaz y sencillo de las listas o tablas. Examinemos detalladamente este importante aspecto del LOGO para estudiar después sus posibles aplicaciones.

Es oportuno aclarar, ante todo, el concepto de lista o tabla: con este término se indica un conjunto de elementos que se pueden expresar de forma tabular. En el interior de una pareja de parén-

tesis cuadrados (la posición que ocupa un elemento en el ámbito de una tabla debe de ser contada a partir del extremo izquierdo de la lista).

Por ejemplo, si se quiere asignar al listado "ALFA" el conjunto de los elementos 1,A,3F escribiremos:

MAKE "ALFA (1 A 3F)

Por conjunto de elementos se entiende, por tanto, una colección, un grupo de números y/o palabras, constantes y/o variables.

Es importante subrayar que un listado está vacío (()) si no contiene elementos o, mejor dicho, contiene exclusivamente el elemento nulo, propiamente llamado CERO.

Para definir un listado vacío, la instrucción necesaria es del tipo:

MAKE "NOMBRELISTA ()

Examinemos una por una las primitivas dedicadas al tratamiento de los listados.

[...] = [...]

El operador "=" confronta el contenido de la primera lista con el de la segunda, *entendiendo por contenido el conjunto de los elementos de cada uno en su orden respectivo.* En otros términos, el listado [a b c] es distinto [o no igual] del listado (c a b): aunque contengan los mismos elementos los dos están caracterizados por un orden distinto de tabulación.

Si nosotros pulsamos de manera directa:

[ALFA BETA GAMMA]=[ALFA BETA GAMMA]

la computadora contestará:

RESULT : TRUE

TRUE (=es verdad) confirma la veracidad de la igualdad.

Si, por el contrario, escribimos

[ALFA BETA GAMMA]=[EPSILON GAMMA]

la contestación que debemos esperar será:

RESULT : FALSE

ya que la igualdad, como claramente pone en evidencia el término inglés, es falsa.

El operador "=" puede ser empleado en la comparación de listas y para testimoniar la igualdad entre dos variables o entre una constante y una variable.

Es lícito escribir, por ejemplo, comandos del tipo:

PRINT 4 = 2 * 2

o también

MAKE "X 2 PRINT 4 = :X * :X + 1

que dan como resultados TRUE y FALSE, respectivamente.

Por lo que se refiere a los operadores "menor", "mayor" y "distinto" valen las consideraciones usadas en relación a la primitiva "=".

SENTENCE argumento 1 argumento 2

Esta instrucción sirve para unir, en una lista única, los contenidos de las listas especificadas (argumento 1 + argumento 2).

Por ejemplo:

PRINT SENTENCE [HASTA PRONTO] [QUERIDO AMIGO]

determina la impresión de "HASTA PRONTO QUERIDO AMIGO".

Como el operador "=" también la primitiva SENTENCE puede trabajar con listas y con constantes o variables numéricas, alfanuméricas o alfabéticas.

Así están permitidas instrucciones del tipo:

SENTENCE "ADIOS [QUERIDO AMIGO]

(constante alfabética+listado=lista) o

SENTENCE :NOMBRE [LISTA ELEMENTOS]

(variable alfanumérica+listado=lista) o también:

SENTENCE 14 "Es un número par

(constante numérica+constante alfanumérica=lista).

FIRST argumento

El argumento de esta primitiva puede ser indiferentemente una lista, una variable o una constante.

La instrucción FIRST devuelve el primer elemento del argumento Por ejemplo:

PRINT FIRST [ABC DEF GHI]

imprime en la pantalla "ABC".

Y también

PRINT FIRST "nos vemos

determinará la letra "n".

Tenga en cuenta qué cualquiera que sea el argumento de FIRST (un número, una variable o una lista) esta primitiva genera como salida siempre una lista.

Haciendo referencias a los ejemplos anteriores podemos estar seguros que "ABC" y "N" son dos listas. Si planteamos, por lo tanto, las siguientes igualdades:

[ABC]=FIRST [ABC DEF GHI]
[A]=FIRST "NOS VEMOS

tendremos con seguridad dos condiciones "TRUE".

Hay que tener presente, en la fase de programación, que si el argumento de la primitiva FIRST es nulo (first [], first") se generará un error, con la consecuente interrupción del procedimiento que se estaba llevando a cabo.

LAST argumento

Esta instrucción desarrolla la función opuesta al mando FIRST: devuelve el último de los elementos que constituyen el argumento, que puede ser de nuevo, indiferentemente, una lista, una variable (numérica, alfanumérica o alfabética), o una constante (numérica, alfanumérica o alfabética).

Utilizando los ejemplos usados anteriormente para aclarar el concepto de la primitiva FIRST, podemos ahora escribir:

PRINT LAST [ABC DEF GHI]

o también

PRINT LAST "NOS VEMOS

La computadora contestará respectivamente : "GHI" y "S".

También para la primitiva LAST valen las observaciones hechas anteriormente al examinar el comando "FIRST": de cualquier tipo (lista, variable...) que sea el argumento, ejecutando esta primitiva tendremos siempre como resultado una lista.

BUTFIRST argumento

BUTFIRST, término que puede ser traducido al español con la expresión "excepto el primero", genera como salida un valor igual al argumento, pero sin su primer elemento.

Pulsando de manera directa:

[ABC 123]=BUTFIRST [DEF ABC 123]

o

[MIGO]=BUTFIRST "AMIGO

tendremos como resultado dos contestaciones del tipo:

RESULT : TRUE

BUTLAST argumento

Lleva a cabo la función contraria a la primitiva BUTFIRST.

Recurriendo a la instrucción BUTLAST (=excepto el último) se genera como salida un valor igual al argumento, pero sin su primer elemento.

Para tener un ejemplo de lo dicho anteriormente pulsemos

PRINT BUTLAST [1 2 B C 3 4]

y también

PRINT BUTLAST "QUERIDISIMO

y tendremos, respectivamente, la impresión de "1 2 B C 3" y "QUERIDISIMO".

LIST argumento 1 argumento 2

Esta es una instrucción básica dedicada al tratamiento de las listas.

Requiere como entrada dos argumentos; la salida estará constituida por una nueva lista que contendrá en sucesión los dos argumentos. Veamos un ejemplo que aclare este concepto.

Pulsemos de manera directa

LIST [A B] [C D]

La computadora contestará con:

RESULT : [[A B] [C D]]

O sea, con una lista derivada de la unión de los dos argumentos; ha sido generada una "lista de dos listas".

De hecho [[A B] [C D]] está formada por los listados [A B] y [C D] a su vez constituidos por elementos (a, b) y (c, d).

Para entender a fondo la sustancial diferencia existente entre la instrucción LIST y la aparentemente igual SENTENCE observe atentamente el ejemplo que sigue:

PRINT FIRST SENTENCE [A B] [C D]

genera

RESULT : A

PRINT FIRST LIST [A B] [C D]

imprime

RESULT : [A B]

Como se ha dicho para otras primitivas de manipulación de listas, la instrucción LIST admite también argumentos numéricos (ejemplo: LIST 123 456 = [123] [456]) o variables.

FPUT argumento 1 argumento 2

Esta instrucción exige que el argumento 2 sea necesariamente una lista.

Antes hemos observado cómo la lista generada por la primitiva LIST está constituida por el conjunto de los dos argumentos. La función confiada a la instrucción FPUT es, sin duda, aparentemente similar, pero al examinarla menos superficialmente resulta notablemente distinta.

Escribamos de manera directa:

LIST [A B C] [D E F]

y tendremos

RESULT : [[A B C] [D E F]]

Si pulsamos

FPUT [A B C] [D E F]

obtendremos la impresión de

RESULT : [[A B C] D E F]

es fácil entender que mientras la primitiva LIST crea una lista con dos elementos (que a su vez son listados), la primitiva FPUT devuelve como salida una nueva lista cuyo primer elemento es el ARGUMENTO 1, mientras los restantes miembros son los mismos que constituyen el ARGUMENTO 2.

Haciendo referencia al ejemplo de que hemos hablado anteriormente podremos ahora comprender por qué

PRINT LAST LIST [A B C] [D E F]

dará el mensaje RESULT : [D E F]

mientras que las instrucciones

PRINT LAST FPUT [A B C] [D E F]

dan origen a la lista [F].

LPUT argumento 1 argumento 2

El argumento 1 debe ser necesariamente una lista.

Esta primitiva proporciona como salida una lista constituida por los elementos que forman el ARGUMENTO 1 y que tiene como último miembro al propio ARGUMENTO 2.

Intentemos utilizar esta instrucción:

```
PRINT LPUT [1 2 3] [5 6 7]
PRINT LAST LPUT [1 2 3] [5 6 7]
PRINT FIRST LPUT [1 2 3] [5 6 7]
```

obtendremos, respectivamente, las impresiones:

```
[1 2 3 [5 6 7]]
5 6 7
1
```

Comandos de comprobación

La primitiva LIST? controla que el argumento sea de hecho una lista; en caso afirmativo devuelve el mensaje TRUE y en caso contrario FALSE (falso, el argumento no es una lista).

Tendremos, por lo tanto:

```
LIST? [a b c] → RESULT : TRUE
LIST? "frase  → RESULT : FALSE
LIST? 134     → RESULT : FALSE
```

Análoga es la función confiada a la primitiva WORD?: comprueba que el argumento es una variable; en ese caso la salida es TRUE, y si es lo contrario devuelve el mensaje FALSE.

En correspondencia con el ejemplo anterior obtendremos:

```
WORD? [a b c] → RESULT : FALSO
WORD? "frase  → RESULT : TRUE
WORD? 123     → RESULT : TRUE
```

Examinemos finalmente la primitiva EMPTY? La sintaxis general es:

```
PRINT EMPTY? :NOMBRELISTA
```

o también

```
PRINT EMPTY? :NOMBREVARIABLE
```

Esta instrucción examina el contenido del argumento: si es nulo genera el mensaje TRUE, y si no, devuelve el valor FALSE.

Programa resumen

Seguidamente les presentamos un listado del programa REPLAY, que ilustra las posibles aplicaciones en la programación de las listas.

El programa trabaja en alta resolución; permite inicialmente al operador desplazarse a través de la presión de las teclas "A" (=adelante), "B" (=atrás), "I" (=giro a la izquierda) y "D" (=giro a la derecha) con lo cual la tortuga va dejando una pista en la pantalla.

Esta es la fase que se podría definir como "pasiva" del programa: éste memoriza en una lista todos los desplazamientos ordenados por el usuario.

Cuando hayamos dibujado todo lo que nos habíamos propuesto pulsemos la tecla "R" (replay) para poner en marcha la segunda parte del programa: se cancela la página gráfica y la tortuga, por sí sola, lleva a cabo la misma figura anteriormente trazada por nosotros.

Esto ha sido posible gracias a la gran potencia del lenguaje: ha sido suficiente volver a llevar a cabo (RUN) una a una todas las informaciones de nuestros desplazamientos que anteriormente habían sido memorizados en una lista.

El listado del programa es el siguiente (los números de línea que aparecen se han puesto simplemente a efectos de facilitar la posterior explicación; por lo tanto NO deben editarse):

```
01 TO REPLAY
02  MAKE LISTA.COMANDOS [] ;[BORRA LA
     LISTA QUE CONTENDRA TODOS NUESTROS
     MOVIMIENTOS]
03  HOME ;[CENTRA LA TORTUGA]
04  CLEARSCREEN ;[LIMPIA LA PANTALLA]
05  PRINT [PULSAR A, B, D, I (F = FINAL )
     O BIEN R (REPLAY)]
06  GET
07 END
08 TO GET
09  MAKE "TECLAPULSADA READCHARACTER
10  IF :TECLAPULSADA = "R THEN HOME
    CLEARSCREEN EJECUTA.LISTA.COMANDOS
```

```
      NODRAW PRINT [RETORNO] STOP
11  IF TECLAPULSADA = "A THEN PRINT
      [ADELANTE] FORWARD 10
      MAKE "LISTA.COMANDOS SENTENCE
      :LISTA.COMANDOS [FORWARD 10]
12  IF TECLAPULSADA = "B THEN PRINT
      [ATRAS] BACK 10
      MAKE "LISTA.COMANDOS SENTENCE
      :LISTA.COMANDOS [BACK 10]
13  IF TECLAPULSADA = "D THEN PRINT
      [DERECHA] RIGHT 15
      MAKE "LISTA.COMANDOS SENTENCE
      :LISTA.COMANDOS [RIGHT 15]
14  IF TECLAPULSADA = "I THEN PRINT
      [IZQUIERDA] LEFT 15
      MAKE "LISTA.COMANDOS SENTENCE
      :LISTA.COMANDOS [LEFT 15]
15  GET
16 END
17 TO EJECUTA.LISTA.COMANDOS
18  RUN SENTENCE FIRST :LISTA.COMANDOS
      FIRST  BUTFIRST :LISTA.COMANDOS
19  MAKE "LIST.COMANDOS  BUTFIRST
      BUTFIRST :LISTA.COMANDOS
20  IF :LISTA.COMANDOS = [] THEN REPLAY
```



```
21  EJECUTA.LISTA.COMANDOS
22 END
```

Intentemos entender a fondo el funcionamiento del programa REPLAY que proporciona un anticipado y breve análisis, por lo que se refiere a la utilización de la primitiva IF, de la que hablaremos con más profundidad en el próximo capítulo.

- LINEAS 01, 02, 03 y 04
  Autoexplicativas.
- LINEA 05
  Imprime en la pantalla el mensaje que advierte al usuario cuáles son las teclas para el movimiento y en qué dirección generan el desplazamiento de la tortuga.
- LINEA 06
  Salto incondicionado al subprocedimiento GET que empieza (TO GET) en la línea 08.
- LINEA 09
  La instrucción READCHARACTER detiene la ejecución del programa hasta que el operador pulse una tecla.
  La primitiva MAKE asigna al argumento la letra correspondiente al desplazamiento deseado.
  Supongamos ahora que nuestra elección haya sido la D (giro a la derecha).
  El programa lleva a cabo los tests de las líneas 10, 11, 12, pero los encontrarán todos falsos. Llega, por lo tanto, a la línea 13, que realiza el giro hacia la derecha.
  El IF comprueba que la tecla pulsada por nosotros es, efectivamente, "D". Se imprime, por lo tanto, el mensaje (DERECHA) y después la tortuga gira 15 grados a la derecha (RIGHT 15).
  En esta fase del procedimiento GET nuestro movimiento se memoriza en la lista: esta función está confiada a las instrucciones:

  MAKE "LISTA.COMANDOS SENTENCE :LISTA.COMANDOS [RIGHT 15]

  En cuanto se hayan llevado a cabo las instrucciones que siguen al IF se vuelve a la ejecución normal del programa: la línea 14 se lleva a cabo; como el test es falso (nuestra elección es "D" y no "I") se pasa a la línea 15.
  En este momento vuelve a aparecer el procedimiento GET y el ciclo se repite. Para las líneas 11, 12, 14 valen las mismas observaciones realizadas para la línea 13.

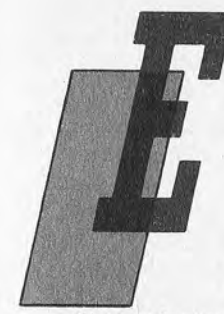
Si en un cierto punto el usuario pulsa la tecla "R" (REPLAY) el test de la línea 10 toma el valor TRUE y las instrucciones siguientes, por lo tanto, sí se efectúan. Se centra la tortuga y se limpia la página de alta resolución. Después se ejecuta el subprocedimiento EJECUTA.LISTA.COMANDOS, cuyo inicio está en la línea 17.
Supongamos que en el momento de esta llamada el :LISTA.COMANDOS tiene el valor:

[RIGHT 15 FORWARD 10 LEFT 15 FORWARD 10 RIGHT 15]

- LINEA 18
  La instrucción RUN ejecuta el comando presente en el argumento.
  Pero este argumento no es otra cosa que el conjunto (SENTENCE) de los primeros dos elementos del listado en el supuesto (RIGHT 15), obtenidos con SENTENCE de FIRST y FIRST BUTFIRST, que indican cuál había sido nuestro primer desplazamiento.
- LINEA 19
  De la lista de los desplazamientos se elimina la codificación de la primera instrucción (se le quitan los primeros dos elementos).
- LINEA 20
  Si LISTA.COMANDOS ha sido llevado a cabo de manera completa, el programa REPLAY empieza de nuevo.
- LINEA 21
  En caso contrario se efectúa la ejecución del desplazamiento siguiente.

CAPITULO IV

BUCLES, COMPARACIONES Y RECURSIVIDAD



En el segundo y tercer capítulos hemos visto cómo el intérprete LOGO permite la repetida ejecución de grupos específicos de instrucciones.

Cuando necesitamos que un conjunto de mandos sea ejecutado un número "N" de veces podríamos resolver el problema (muy poco elegantemente) a través de la sucesiva definición de procedimientos.

Sin embargo, con el LOGO tenemos la posibilidad de crear macro-instrucciones que lleven a cabo una función un determinado número de veces que nosotros especificamos a priori en el momento de diseñar el programa.

Muy a menudo es también necesario poder llamar a comandos particulares un número de veces variable según los casos. En este capítulo podremos constatar y estudiar cómo el intérprete LOGO nos permite resolver con sencillez y potencia el problema que tratamos.

En la segunda parte del capítulo se examinarán las principales primitivas que nos permiten tomar decisiones, mientras la última parte la hemos reservado para una fabulosa herramienta, que permite obtener resultados importantes y que se conoce con el nombre de recursividad.

La instrucción "REPEAT"

Escribamos el programa listado más abajo: ordenando su ejecución nos pide que especifiquemos cuál es el carácter que que-

remos imprimir en la pantalla y el número de veces que deseamos que se repita la fase de impresión:

```
TO INPUTNUMERO

  PRINT [ESCRIBE CUANTAS VECES]
  PRINT []
  PRINT [DEBO REPETIR LA ACCION]
  PRINT []
  PRINT [DE IMPRESION]
  PRINT []
  PRINT [Y LUEGO PULSA <RETURN>]
  PRINT []
  MAKE "NUMERO REQUEST
  OUTPUT FIRST :NUMERO

END

TO INPUTCARACTER

  PRINT [PULSA EL CARACTER]
  PRINT []
  PRINT [A IMPRIMIR]
  MAKE "CARACTER READCHARACTER
  PRINT []
  PRINT :CARACTER
  PRINT []

END

TO PROGRAMA.REPETIR

  MAKE "CARACTER
  INPUTCARACTER
  MAKE "CICLOS INPUTNUMERO
  REPEAT :CICLOS [PRINT1 :CARACTER]
  PRINT []

END
```


El funcionamiento del programa debería estar claro, excepto el recurso, en la antepenúltima línea, a la instrucción:

REPEAT :CICLOS [PRINT 1 :CARACTER]

La primitiva REPEAT efectúa un grupo de instrucciones (Procedimientos y/o primitivas) el número de veces especificadas por el argumento.

De hecho, la sintaxis necesaria para la utilización de la instrucción REPEAT requiere:

1. Un número colocado inmediatamente después de la palabra clave (en el ejemplo es la variable numérica :CICLOS) que especifica el número de veces que deberán repetirse las instrucciones siguientes.
2. Una o más instrucciones, siempre cerradas entre una pareja de paréntesis cuadrados.

En otras palabras, podemos repetir la ejecución de unos subprocedimientos, unas sencillas primitivas como FORWARD, RIGHT, BACK..., una serie de instrucciones o bien nada menos que varios subprocedimientos encadenados.

Intentemos ahora analizar su funcionamiento a través de la ayuda ofrecida por una serie de programas de ejemplo.

Dibujemos un rectángulo (recordemos cómo habíamos hecho esto en el capítulo tercero, con un gran gasto de memoria):

TO RECTANGULO

Llamemos a este procedimiento RECTANGULO predisponiendo así al intérprete LOGO para que lo acepte como una palabra nueva del vocabulario; después pulsemos

REPEAT 2 [FORWARD 30 LEFT 90 FORWARD 50 LEFT 90]

La instrucción REPEAT tiene como parámetro el número de veces que la lista de instrucciones debe repetirse (2 veces en nuestro caso) y estas mismas que, en orden, son: FORWARD 30, LEFT 90, FORWARD 50, LEFT 90.

Lancemos ahora el programa y tendremos lo que se muestra en la figura 1.

Existe un método aún más correcto para obtener el mismo resultado:

Definir un subprocedimiento del tipo:

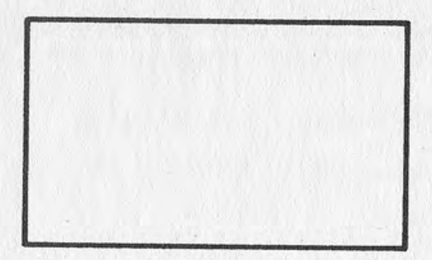


Figura 1.—Rectángulo obtenido mediante la instrucción REPEAT.

```
TO ARISTA

  FORWARD 30
  LEFT 90
  FORWARD 50
  LEFT 90

END
```

y luego llamarlo de esta manera:

```
TO RECTANGULO

  REPEAT 2 [ARISTA]

END
```

Intentemos ahora construir otro procedimiento que utilice REPEAT y a la vez el subprocedimiento RECTANGULO:

```
TO VENTANA

  REPEAT 2 [RECTANGULO FORWARD 30]
  LEFT 90
  FORWARD 25
  LEFT 90
  FORWARD 60

END
```


Obtendremos lo que se ve en la figura 2.

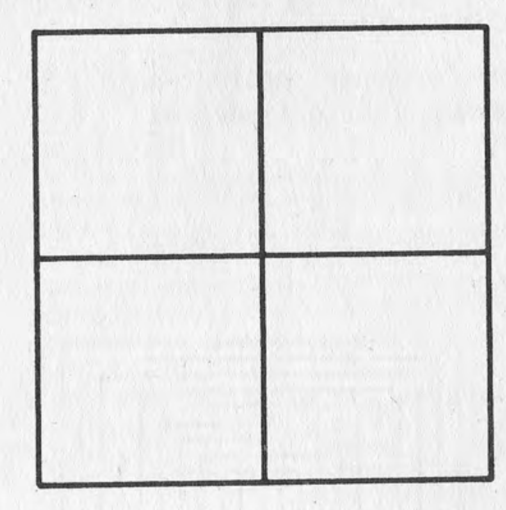


Figura 2.—Llamadas repetidas a RECTANGULO.

Ahora veamos cómo conseguir una estrella (Fig. 3):

```
TO ESTRELLA

  REPEAT 18 [FORWARD 100 RIGHT 140]

END
```

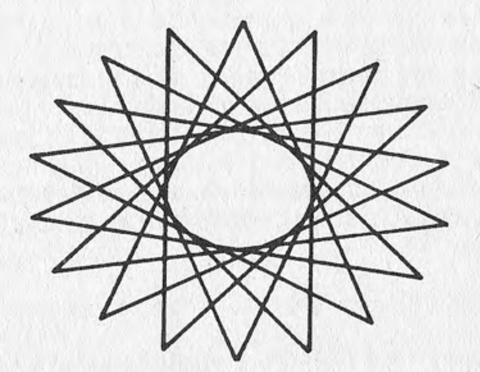


Figura 3.—Otras posibilidades de la instrucción REPEAT.

Y, para acabar, una espiral:

```
TO ESPIRAL  :RADIO

  REPEAT 30 [MAKE "RADIO :RADIO + 5
    FORWARD :RADIO RIGHT 90]

END
```

Para ver los resultados (Fig. 4) pulsar:

ESPIRAL 10

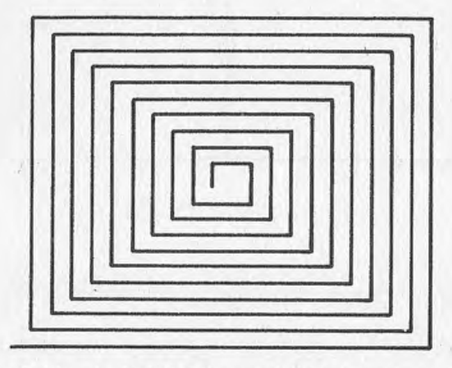


Figura 4.—Unión de REPEAT y variables de entrada.

En el procedimiento ESPIRAL aparece otra vez, como verán, la utilización de una variable. Estas son las artífices de una programación eficaz, ya que, permitiendo la variación en el interior de un ciclo de los parámetros esenciales, nos dejan cambiar notablemente las características de lo que hacemos sin tener que recurrir a la utilización de nuevas reescrituras del programa.

Hemos visto ya en el capítulo anterior la utilización de las variables en los procedimientos y subprocedimientos; veamos ahora cómo "explotarlas" mejor en relación a la instrucción REPEAT.

Sea CUADRADO un procedimiento sencillo que trace la figura homónima:

```
TO CUADRADO  :LADO

  REPEAT 4 [FORWARD :LADO RIGHT 90]

END
```


Para ejecutarlo hará falta dar como entrada no sólo el nombre del procedimiento, sino también el valor inicial de las variables asociadas. En otras palabras: si deseamos que el cuadrado tenga como lado 10.. 20.. 30.. deberemos escribir:

RUN CUADRADO 10
RUN CUADRADO 20

y así seguidamente (en el capítulo 3 debería haber quedado claro este concepto).

De esta manera es sencillo trazar figuras similares con un solo programa en que haya sido introducida de manera oportuna una variable y, eventualmente, construir un procedimiento de control con los decrementos o incrementos deseados (como hicimos en ESPIRAL).

De forma particular veamos cómo incrementar una variable en un ciclo dominado por REPEAT para dibujar una serie de figuras "concéntricas" (Fig. 5):

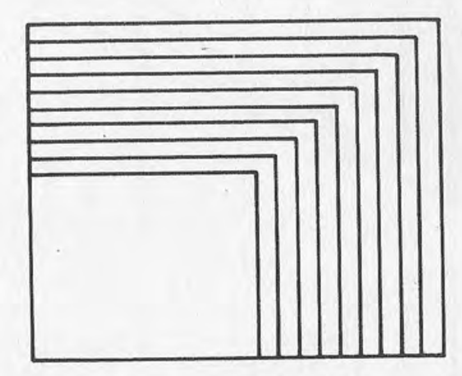


Figura 5.—Combinación de REPEAT y MAKE con variables.

La instrucción MAKE, como ya sabemos, permite efectuar operaciones con las variables (en nuestro caso, una suma) durante la ejecución del programa.

```
TO CUADRADO  :LADO

  REPEAT 4 [FORWARD :LADO RIGHT 90]

END
```

```
TO FIGURAS.CONCENTRICAS :DIAMETRO

  REPEAT 10 [MAKE "DIAMETRO :DIAMETRO+10
     CUADRADO :DIAMETRO]

END
```

Finalmente, estudiemos una interesante serie de procedimientos que están en correlación y que dibujan los triángulos de la figura 6.

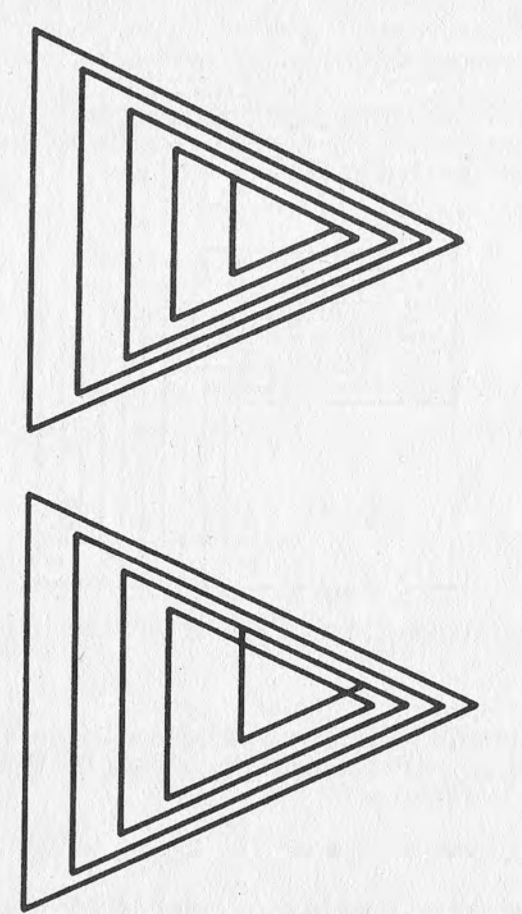


Figura 6.—Dos ejemplos del procedimiento CINCO.RECTANGULOS.

Pulsando, por ejemplo, RUN CINCOTRIANGULOS 10 y RUN CINCOTRIANGULOS 30 tendremos, sucesivamente, las figuras:

```
TO TRIANGULO :LADO

  REPEAT 3 [FORWARD  :LADO RIGHT 120]

END


TO CINCO.TRIANGULOS :LADO

  REPEAT 5 [MAKE "LADO :LADO + 40
     TRIANGULO :LADO   ALFA :LADO]

END


TO ALFA :LADO

  PENUP
  HOME
  LEFT 90
  FORWARD :LADO / 2
  LEFT 90
  FORWARD :LADO / 2
  RIGHT 180
  PENDOWN

END
```

Comparaciones y bifurcaciones

El lenguaje LOGO permite al programador efectuar controles de tipo decisivo sobre los valores de las variables, tanto numéricas como literales.

Para comprender cualitativamente cuál es el significado y la función de las instrucciones de test, tomemos en consideración

un ejemplo, aparentemente banal, que encierra en sí, al examinarlo menos superficialmente, varias ocasiones de reflexión.

Supongamos que viajamos por una autopista con nuestro coche; en cierto momento del recorrido se nos presenta a la vista una bifurcación de carretera con sus correspondientes carteles avisadores (Fig. 7). Puesto que para nosotros está claro cuál es la meta de nuestro recorrido, nos vemos obligados a efectuar una elección de la dirección y tomar, en consecuencia, el camino oportuno.

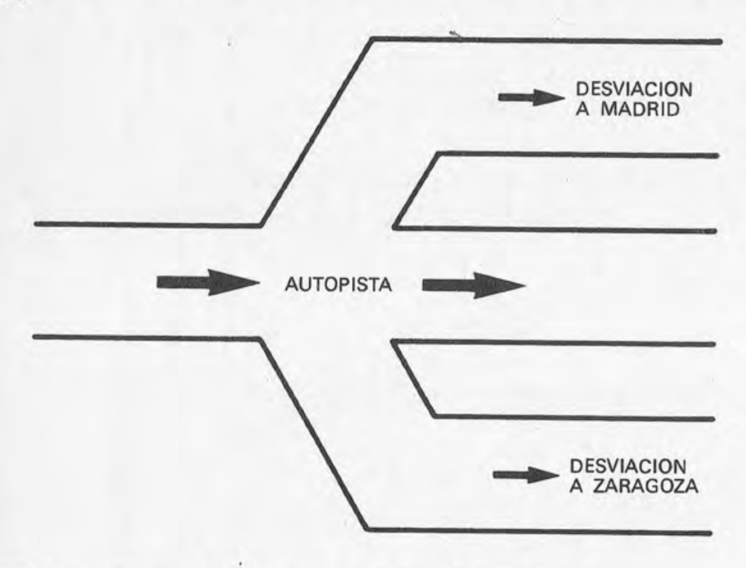


Figura 7.—Una simple disyuntiva en el camino.

Si nuestra meta es, por ejemplo, la ciudad de Madrid, entre las dos direcciones (que llevan una a Zaragoza y la otra a Madrid) elegiremos la segunda alternativa.

¿Qué operaciones ha hecho nuestra mente en el momento en que hemos tomado la decisión? El proceso habrá sido, más o menos, el siguiente:

- SI la dirección que quiero (Madrid)=indicaciones de la carretera, ENTONCES tomo la desviación correspondiente.
- SI la dirección que quiero (Madrid) NO=indicaciones de la carretera, ENTONCES sigo derecho.

En otros términos: hemos confrontado lo que queremos con lo que vemos.

De la misma manera actúa el intérprete LOGO en el momento de tomar una decisión.

Si deseamos que cierto procedimiento o subprocedimiento se interrumpa en un cierto momento solamente si se han verificado determinadas condiciones, por ejemplo en el momento en que la variable A toma el valor 100, tendremos que escribir:

```
IF :A=100 STOP
```

En consecuencia, cada vez que el procedimiento llegue a esta instrucción se operará una comparación entre el valor real y la constante 100; sólo en el caso en que la igualdad sea verdadera (:A=100) se efectuará la instrucción STOP (o cualquier otro comando especificado).

En el ejemplo anterior ha sido utilizado el operador de igualdad ("="), pero existen otros que nos permiten una casuística muy completa.

Presentamos seguidamente en la tabla 1 los operadores de relación y las funciones que llevan a cabo.

OPERADOR	SIGNIFICADO
=	IGUAL
>	MAYOR
<	MENOR
>=	MAYOR O IGUAL
<=	MENOR O IGUAL
<>	DISTINTOS

Tabla 1.—Operadores de relación y comparaciones que desempeñan.

Volviendo a hablar de la utilización de la instrucción IF, su forma más sencilla es:

IF (expresión booleana) THEN (instrucción-es-)

Intentemos ahora analizar detalladamente la expresión:

1. La primitiva "IF" advierte al intérprete LOGO que se debe efectuar una comparación.
2. Por expresión booleana se entiende cualquier tipo de comparación válida (a=b, a<>b, a>b, a<b,....); el término "booleana" toma su nombre del matemático George Boole. El resultado de una expresión de Boole puede tomar exclusivamente los valores verdadero o falso (TRUE o FALSE); por ejemplo, si a=1 y b=1, entonces a=b es "verdadero", pero si a=1 y b=2, a=b es "falso".
3. El comando "THEN" indica que se desarrollen las operaciones que siguen inmediatamente después exclusivamente en el caso de que el resultado de la expresión booleana sea "verdadero".
4. El término (instrucción-es-) se sustituirá por la serie de operaciones que nosotros queramos que se lleven a cabo en el caso de que la expresión booleana sea verdadera.

Aclaremos estos conceptos por medio del siguiente ejemplo (programa "INTERSECCIONES"). Supongamos que queremos girar una figura hasta el momento en que un contador supere un determinado valor numérico que nosotros elegimos igual a 100.

```
TO INTERSECCIONES :CONTADOR :GIRO

  IF :CONTADOR > 100 THEN PRINT "FINAL
    DE LA EJECUCION  STOP
  ROTACION :GIRO
  INTERSECCIONES (:CONTADOR+10) :GIRO

END


TO ROTACION :GIRO

  CUADRADO
  LEFT :GIRO

END
```



```
TO CUADRADO

  REPEAT 4 [FORWARD 60 RIGHT 90]

END
```

El programa llama sucesivamente a tres subprocedimientos (INTERSECCIONES, ROTACION y CUADRADO) que trazan una serie de cuadrados que al girar unos respecto a los otros generan un efecto que se observa en la figura 8.

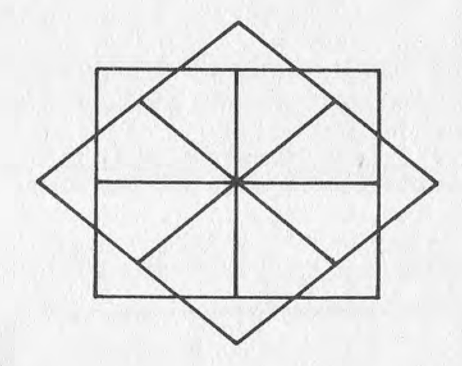


Figura 8.—Efecto conseguido al dibujar varios cuadrados girados.

La instrucción

IF CONTADOR > 100 THEN PRINT "FINAL DE LA EJECUCION STOP

para el programa, que de otra manera seguiría hasta el infinito.
Estudiemos otro ejemplo:

```
TO LLAMADA :PASO :LONGITUD

  LEFT 90
  DIBUJA :PASO :LONGITUD

END
```

```
TO DIBUJA :PASO :LONGITUD

   FORWARD 1
   MAKE "LONGITUD :LONGITUD - 1 - :PASO
   PENUP
   FORWARD :PASO
   PENDOWN
   IF :LONGITUD < 1 THEN STOP
   DIBUJA :PASO :LONGITUD

END
```

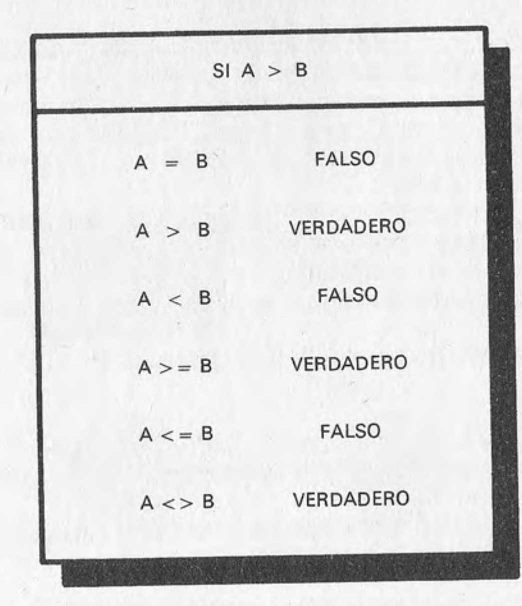
El programa "DIBUJA :PASO :LONGITUD" exige como datos el paso del gráfico (la distancia entre una línea y la siguiente) y la longitud total del dibujo.

Gracias a la utilización del operador de relaciones "<" la ejecución del procedimiento termina en el momento en que la medida del gráfico a trazar toma un valor inferior a 1.

Más abajo presentamos tres tablas (2, 3 y 4) que muestran las combinaciones posibles entre los fundamentales operadores de

SI A = B	
A = B	VERDADERO
A > B	FALSO
A < B	FALSO
A >= B	VERDADERO
A <= B	VERDADERO
A <> B	FALSO

Tabla 2.—Resultados posibles supuesto A=B.



SI A > B	
A = B	FALSO
A > B	VERDADERO
A < B	FALSO
A >= B	VERDADERO
A <= B	FALSO
A <> B	VERDADERO

Tabla 3.—Partiendo de A>B los resultados de combinar los operadores lógicos de relación son los que se muestran.

SI A < B	
A = B	FALSO
A > B	FALSO
A < B	VERDADERO
A >= B	FALSO
A <= B	VERDADERO
A <> B	VERDADERO

Tabla 4.—La última hipótesis es A<B.

relaciones y los resultados de aplicarlos a dos variables; la primera línea de cada tabla constituye la hipótesis en base a la cual (a través de los mismos operadores) se llega a obtener un resultado de tipo booleano (verdadero o falso). Por ejemplo, en la tabla 2, donde suponemos A=B, la relación A>B será falsa, en tanto en la tabla 3 será verdadera.

Una vez entendidos los fundamentos de la instrucción IF podemos pasar ahora a ver otra expresión posible.

Escribámosla en la siguiente forma e intentemos comprender las diferencias fundamentales con lo anteriormente dicho:

IF (expresión booleana) THEN (Instrucción 1) ELSE (instrucción 2)

- Si la expresión booleana es verdadera (vale TRUE) se llevará a cabo entonces la instrucción 1 y se ignorará la instrucción 2;
- Si la expresión booleana es falsa (vale FALSE) se llevará a cabo entonces la instrucción 2 y se ignorará la 1.

Es evidente que el grupo IF... THEN... ELSE permite una mayor y mejor estructuración del programa.

En la figura 9 encontramos el diagrama de flujo para la instrucción IF... THEN... ELSE.

El procedimiento que sigue demuestra una aplicación posible de la instrucción "ELSE".

El programa pregunta dos números e imprime un mensaje según su relación (que el primero sea mayor que el segundo, los dos iguales o el primero es menor que el segundo).

```
TO PROG

  CLEARTEXT
  INSTRUCCIONES ;[ESCRIBE LAS
    INSTRUCCIONES PARA EL USUARIO]
  MAKE "X INPUTNUMERO ;[PRIMER NUMERO]
  INSTRUCCIONES
  MAKE "Y INPUTNUMERO ;[SEGUNDO NUMERO]
  IF :X = :Y THEN CASO.IGUALES ELSE IF
    :X > :Y THEN CASO.MAYOR ELSE
    CASO.MENOR

END
```

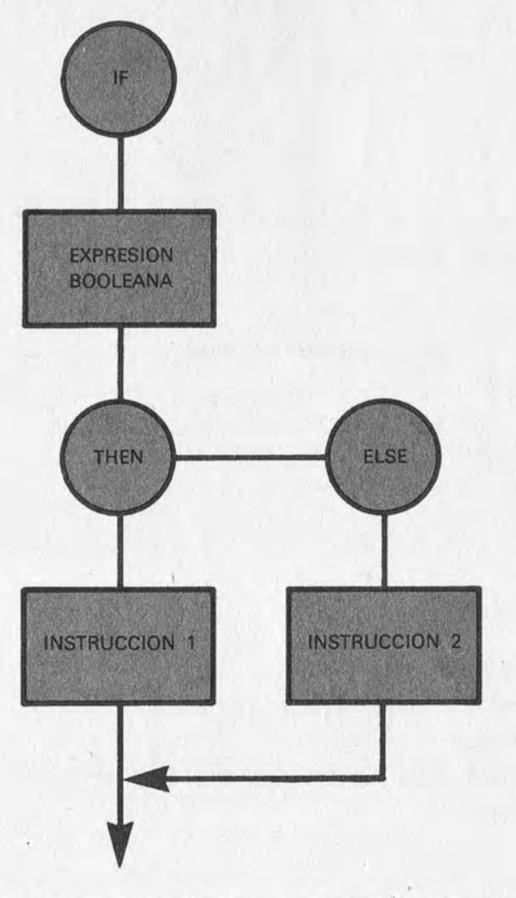



Figura 9.—Diagrama de bloques de la instrucción IF... THEN... ELSE.

```
TO INPUTNUMERO

  OUTPUT FIRST (REQUEST)

END


TO CASO.IGUALES

  PRINT "
  PRINT [EL PRIMER NUMERO]
```

```
   PRINT "
   PRINT [ES IGUAL AL SEGUNDO]
   PRINT "

END


TO CASO.MENOR

   PRINT "
   PRINT [EL PRIMER NUMERO]
   PRINT "
   PRINT [ES MENOR QUE EL SEGUNDO]
   PRINT "

END


TO CASO.MAYOR

   PRINT "
   PRINT [[EL PRIMER NUMERO]
   PRINT "
   PRINT [ES MAYOR QUE EL SEGUNDO]
   PRINT "

END


TO INSTRUCCIONES

   PRINT "
   PRINT [ESCRIBE UN NUMERO]
   PRINT "
   PRINT [Y LUEGO PULSA LA TECLA DE]
   PRINT "
   PRINT [RETURN]
   PRINT "

END
```


Sería un buen ejercicio que intentaran ustedes mejorar los subprocedimientos de salida de mensajes aprovechando la gran zona común que tienen.

TEST... IFTRUE... THEN... ELSE

Supongamos que debemos calcular una sencilla suma y que queremos llamar a un subprocedimiento solamente en el caso particular en que esta suma sea igual a un valor anteriormente fijado.

Para lograr esto podemos recurrir a las instrucciones:

```
TEST :A+:B=:D
IFTRUE THEN SUBPROCEDIMIENTO
```

Aclaremos ahora con la ayuda de un ejemplo (procedimiento VERDADERO.FALSO) cuáles son las distintas posibilidades de la instrucción IFTRUE.

Este procedimiento pide cuatro valores numéricos y arbitrarios (x,y,z,s).

En el caso de que la suma "X+Y+Z" valga "S", el programa imprime el mensaje "es verdadero"; en caso contrario, tendremos como salida "es falso":

```
TO VERDADERO.FALSO

   MAKE "X INPUTNUMERO
   MAKE "Y INPUTNUMERO
   MAKE "Z INPUTNUMERO
   MAKE "S INPUTNUMERO
   MAKE "T  :X + :Y + :Z
   TEST :T = :S
   IFTRUE THEN PRINT [ES VERDADERO] ELSE
     PRINT [ES FALSO]

END


TO INPUTNUMERO

   OUTPUT (REQUEST)

END
```

TEST... IFFALSE... THEN... ELSE

Por lo que se refiere a esta instrucción, el tema es equivalente a IFTRUE, considerando, sin embargo, las hipótesis al contrario.

En otras palabras, el comando TEST verifica si la comparación es verdadera o falsa, pero las instrucciones que siguen a la primitiva THEN se llevan a cabo sólo si el resultado del test resulta ser falso.

El programa anterior, por lo tanto, se modificaría de esta manera:

```
TO VERDADERO.FALSO

  MAKE "X INPUTNUMERO
  MAKE "Y INPUTNUMERO
  MAKE "Z INPUTNUMERO
  MAKE "S INPUTNUMERO
  MAKE "T  :X + :Y + :Z
  TEST :T = :S
  IFFALSE THEN PRINT [ES FALSO] ELSE
    PRINT [ES VERDADERO]

END


TO INPUTNUMERO

  OUTPUT (REQUEST)

END
```

HEADING

Se trata de un comando muy particular. Consiste en una medición de la orientación angular de la plumilla o tortuga (turtle) respecto a su dirección inicial (mirando hacia la parte superior de la pantalla).

Si, por ejemplo, pulsamos:


```
HOME
RIGHT 15
PRINT HEADING
```

obtendremos como contestación

```
15
```

El valor generado por la instrucción HEADING es de tipo numérico y, por lo tanto, es posible confrontarlo con cualquier otro (variable o constante) a través de la acostumbrada relación del tipo:

IF HEADING (Operador relacional) y THEN...

"ANYOF" y "ALLOF"

La sintaxis de estas instrucciones, relativamente sosfisticadas, es:

ANYOF (expresión.booleana 1) (expresión booleana 2)

y

ALLOF (expresión.booleana 1) (expresión booleana 2)

Por lo tanto, ambas instrucciones requieren como parámetros dos expresiones booleanas.

En base a lo anteriormente visto seguramente han intuido que este tipo de relación proporciona un resultado solamente del tipo FALSE o TRUE.

Las dos primitivas en cuestión, según lo que valen los argumentos (expresión 1 y expresión 2), generan el mensaje TRUE o FALSE. La tabla 5 muestra todas las combinaciones posibles.

La primera columna está constituida por resultados de la expresión 1; la segunda, por los relativos a la expresión 2, y la tercera y la cuarta son los valores obtenidos por las primitivas ANYOF y ALLOF, respectivamente. Como verán, equivalen, sencillamente, a los operadores lógicos OR y EXOR.

La recursividad

El término "RECURSIVIDAD" indica el proceso lógico por el cual un proceso en ejecución se llama a su vez.

EXPRES. 1	EXPRES. 2	ANYOF	ALLOF
FALSO	FALSO	FALSO	FALSO
FALSO	TRUE	TRUE	FALSO
TRUE	FALSO	TRUE	FALSO
TRUE	TRUE	TRUE	TRUE

El procedimiento elemental:

```
TO EJEMPLO.RECURSIVIDAD

   FORWARD 50
   LEFT 120
   EJEMPLO.RECURSIVIDAD

END
```

es un claro ejemplo de lo anteriormente dicho.

Es fácil ver que un programa de estas características procede hasta el infinito; no existe, de hecho, ninguna condición para parar la ejecución.

Probablemente el más inmediato y entusiasta campo de aplicación de la técnica de la recursividad es el aspecto gráfico (además de algunos cálculos matemáticos).

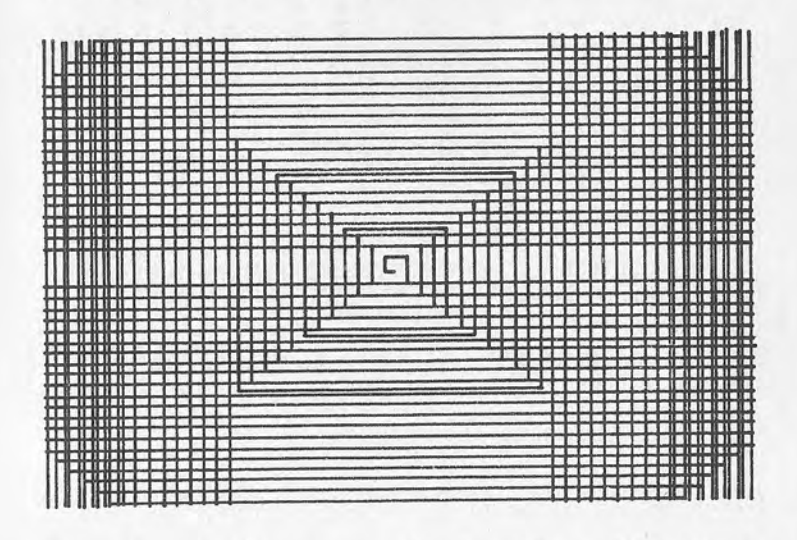
Pulsemos la breve secuencia siguiente (los números anteriores a las instrucciones no deben escribirse; sirven exclusivamente para poder comentar más ágilmente el funcionamiento del programa)

```
01   TO DIBUJA.ESPIRAL
02      FORWARD :LADO
03      RIGHT 90
04      ESPIRAL :LADO+3
05   END
```


y la ejecutamos con

RUN DIBUJA.ESPIRAL 1

La imagen que resulta es la mostrada en la figura 10.



Estudiemos línea por línea el funcionamiento del programa en cuestión.

- LINEA 01
  Está constituida por la declaración del procedimiento ESPIRAL y del parámetro variable :LADO, sobre el cual trabajará.
- LINEA 02
  La instrucción FORWARD :LADO hace que la tortuga vaya hacia adelante un número de posiciones igual al valor de la variable :LADO.
- LINEA 03
  Gira la tortuga 90 grados hacia la izquierda, determinando las aristas de la espiral.
- LINEA 04
  En este momento se efectúa el proceso de recursividad: se llama al procedimiento DIBUJA.ESPIRAL pasándole el valor puesto al día de :LADO (igual a :LADO + 3).

El ciclo empieza de nuevo desde el punto (1) sin terminar nunca.

El lector seguramente habrá intuido que un posible sistema para interrumpir la ejecución de un programa recursivo podría ser recurriendo a las primitivas IF... THEN...

Estudiemos el siguiente programa, un poquito más difícil que los anteriores, que adopta justo esta técnica (recursión + IF..) para calcular un número factorial.

```
01  TO FACT :X
02    FACTORIAL :X :X
03  END

04  TO FACTORIAL :X :Y
05    IF :Y=0 THEN OUTPUT (1)
07    OUTPUT :X * (FACTORIAL :X-1 :Y-1)
08  END
```

Por factorial de un número entero positivo "n" se define matemáticamente el producto de los números enteros sucesivos desde 1 hasta "n".

Por ejemplo, el factorial de 3 es:

1*2*3=6

y el factorial de 6 es:

1*2*3*4*5*6=720

La regla anteriormente descrita admite una excepción: por definición, el factorial de cero es uno (el programa FACT tiene en cuenta también esto).

Para ejecutar el procedimiento se actúa dando el acostumbrado RUN, acompañado por el número del cual queremos que se calcule la función factorial.

Por ejemplo escribamos:

RUN FACT 3

y tendremos el valor (3*2*1):

RESULT : 6

Veamos atentamente cómo funciona el programa (repetiremos de nuevo que las líneas han sido numeradas solamente con el fin de facilitar la búsqueda de las instrucciones; estos números

no forman parte del programa y, por lo tanto, no deben pulsarse) siguiendo el proceso de su ejecución para el ejemplo anterior.

- LINEA 01
  Define el procedimiento FACT; la variable ":X" está colocada, con referencia al ejemplo, en el valor 3.
- LINEA 02
  Se lleva a cabo un salto incondicional hacia el subprocedimiento FACTORIAL, al cual se pasa un doble valor (:X :X).
- LINEA 04
  La variable "X" y la "Y" se ponen al valor numérico 3.
- LINEA 05
  ":Y" vale 3: las instrucciones que siguen al test se ignoran.
- LINEA 06
  Se debe mandar como salida el resultado de X*FACTORIAL de (X-1): De hecho, el factorial de 3 es igual a 3 (=X) por el factorial de 2 (=X-1).
  En este momento el ordenador deberá calcular cuánto vale el factorial de 2, que es la operación que le queda pendiente; llama entonces por recursividad al subprocedimiento FACTORIAL pasándole el valor X-1 (=2) y el Y-1 (=2).
  La ejecución del programa vuelve a la línea 04.
- LINEA 05
  Las instrucciones posteriores al IF se ignoran, ya que el test es TRUE (Y=X=2<>Ø).
- LINEA 06
  El factorial de X (=2) se puede ver como X por el factorial de X-1 (=1); es el segundo nivel de recursividad: se llama otra vez subprocedimiento FACTORIAL pasándole el valor X-1 (=1).
- LINEA 05
  El test no está todavía satisfecho (si han entendido el mecanismo comprenderán que el test será TRUE en el siguiente paso).
- LINEA 06
  El factorial de 1 (X) es 1 por el factorial de 0 (X-1).
  Es el tercer nivel de recursividad; otra vez una llamada al subprocedimiento FACTORIAL con "X" e "Y" que valen, por fin, 0.
- LINEA 05
  El test se verifica: se genera en la salida el valor 1.
  La siguiente instrucción que se lleva a cabo es el END de línea 7.

- LINEA 07
  En este punto el ordenador puede calcular por fin el producto, anteriormente buscado, entre 2 y el factorial de 2-1; el resultado es 2*1=2.
  Este valor permite el postergado cálculo del factorial de 3=3*factorial de dos = 3*2=6.
- LINEA 03
  El programa puede así, finalmente, cerrar su ejecución.
  El lector que haya encontrado algún problema al seguir el hilo del razonamiento, repítalo teniendo en cuenta la ilustración de la figura 10.

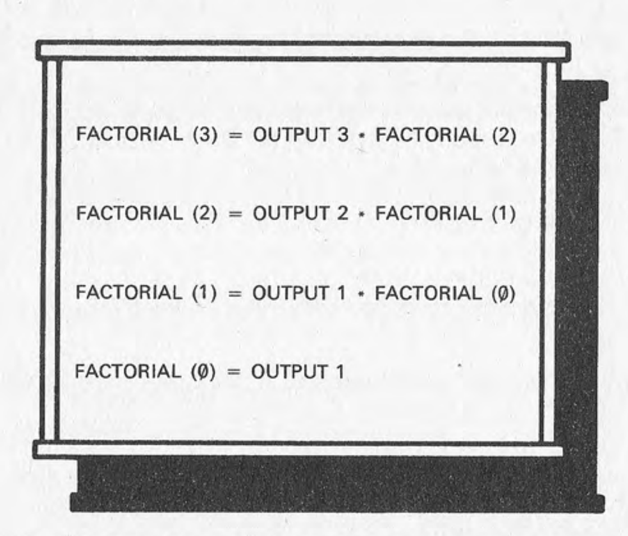


Figura 10.—Esquema para seguir el proceso del cálculo del factorial de 3.

Programas resumen

El programa TRAYECTORIA simula el movimiento de un proyectil del cual se deben proporcionar el valor de la velocidad inicial, el ángulo de inclinación inicial con respecto al eje "x" y el efecto de la gravedad.

```
TO TRAYECTORIA :OMEGA :VELOCIDAD :ALFA
   FORWARD  :VELOCIDAD/10
```



```
   RIGHT :ALFA
   MAKE "ALFA :ALFA+0.9
   IF :ALFA > :OMEGA THEN MAKE "ALFA
     :ALFA-1
   MAKE "VELOCIDAD :VELOCIDAD-1
   IF HEADING > 180 THEN STOP
   TRAYECTORIA :OMEGA :VELOCIDAD :ALFA

END


TO SIMULA :OMEGA :VELOCIDAD :ALFA

   SINGLECOLOR
   HOME
   CLEARSCREEN
   PENUP
   LEFT 90
   FORWARD 130
   RIGHT 180
   LEFT :OMEGA
   PENDOWN
   TRAYECTORIA :OMEGA :VELOCIDAD :ALFA

END
```

Introduciendo los datos 45 50 0 (o sea, RUN SIMULA 45 50 0) se obtiene el resultado de la figura 11.

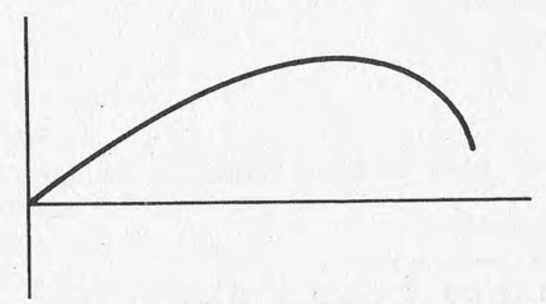


Figura 11.—Trayectoria del proyectil con los valores 45, 50 y 0.

Seguidamente incluimos los listados de algunos procedimientos que sintetizan lo que hemos estudiado hasta ahora:

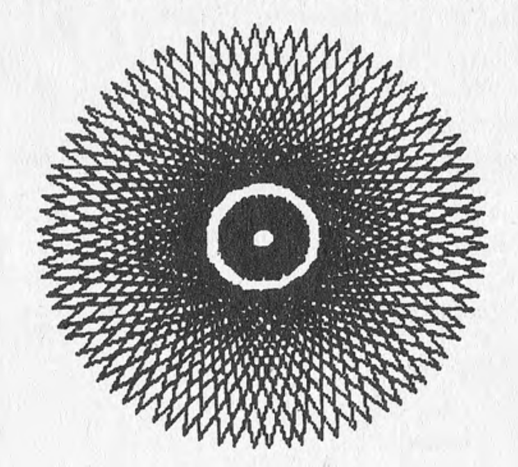


Figura 12.—Una de las figuras obtenidas con estos procedimientos.

Para ejecutarlos utilice el comando RUN acompañado, si es necesario, por los datos de entrada.

PROGRAMA "BORDADO"

```
TO BORDADO

  HOME
  CLEARSCREEN
  ALFA 10 20 30

END

TO ALFA :LADO :ANGULO :PASO

  BACK :LADO
  LEFT :ANGULO
  ALFA :LADO (:ANGULO + :PASO) :PASO

END
```


PROGRAMA "ROSON"

```
TO ROSON
  MAKE "LADO 110
  POLIARISTA :LADO

END

TO POLIARISTA :LADO

  RIGHT :LADO
  REPEAT :LADO/3 [ARISTA :LADO]

END

TO ARISTA :LADO

  FORWARD :LADO
  RIGHT 170
  FORWARD :LADO

END
```

PROGRAMA "INTEGRAL"

```
TO INTEGRAL

  HOME
  CLEARSCREEN
  PENUP
  FORWARD 40
  PENDOWN
  RIGHT 60
  BETA 20

END
```

```
TO BETA :ANGULO

   FORWARD 15
   RIGHT :ANGULO
   BETA :ANGULO+5

END
```

PROGRAMA "PETALOS"

```
TO PETALOS

   HOME
   LEFT 90
   FORWARD 30
   RIGHT 90
   FORWARD 40
   CLEARSCREEN
   REPEAT 12 [RIGHT 120 ARCO 1]

END

TO ARCO :PASO

   FORWARD  :PASO
   RIGHT 6
   IF :PASO < 15 ARCO :PASO+1

END
```

PROGRAMA "FLOR"

```
TO TALLO :X

   BACK 17
   LEFT :X
   IF :X < 20 THEN TALLO :X+2

END
```



```
TO FLOR

   HOME
   FORWARD 30
   CLEARSCREEN
   LEFT 20
   REPEAT 3 [MAKE "CONTADOR 1 PETALO 1]
   RIGHT 40
   TALLO 1

END

TO PETALO :START

   FORWARD 10
   RIGHT :START
   IF :START < 20 THEN PETALO :START+2
   RIGHT 60
   MAKE "CONTADOR :CONTADOR+1
   IF CONTADOR > 2 THEN STOP
   PETALO 1

END
```

CAPITULO V

OPERACIONES NUMERICAS

Representación binaria

Un conjunto de las primitivas del lenguaje está enteramente dedicado a la manipulación de la información numérica binaria.

Vamos a aclarar antes de todo el significado de este término, aunque pueden acudir a volúmenes anteriores de la B.B.I. (especialmente el 1) para más detalles. Sabemos que nuestro ordenador está preparado para resolver una notable cantidad de cálculos, pero no todos conocemos la técnica que la máquina utiliza para representar en su interior la información numérica. A causa de las limitaciones de la electrónica convencional todas las informaciones (también las no numéricas) están codificadas en grupos de bits.

Un BIT es la mínima unidad de información; puede estar exclusivamente en dos estados: activado (nivel 1) o apagado (nivel 0). Mediante la agrupación de bits podemos codificar diversos tipos de informaciones.

Un número decimal, por lo tanto, será representado por el ordenador como una colección de "n" bits.

Estudiemos ahora un algoritmo que nos permita también a nosotros llevar a cabo la conversión desde un valor numérico (para simplificar la cosa, entero y positivo) al correspondiente equivalente binario.

Tomemos como ejemplo el número decimal 131. El proceso de conversión será:

Dividirlo por 2.

131/2=cociente 65 resto 1

Dividamos de nuevo el cociente que hemos obtenido por 2:

65/2=cociente 32 resto 1

Procedamos igual hasta obtener un cociente CERO.

32/2=cociente 16 resto 0
16/2=cociente 8 resto 0
8/2=cociente 4 resto 0
4/2=cociente 2 resto 0
2/2=cociente 1 resto 0
1/2=cociente 0 resto 1

El cociente es nulo y, por lo tanto, podemos terminar las operaciones.

En este momento transcribimos en orden inverso (desde el final al principio) los restos de las sucesivas divisiones, obteniendo:

1 0 0 0 0 0 1 1

Pues bien, ésta es la representación binaria del número decimal 131.

Operadores lógicos binarios

El lenguaje LOGO, como hemos dicho anteriormente, dispone de instrucciones preparadas para el tratamiento de la información numérica binaria; estudiémoslas atentamente.

BITAND argumento 1 argumento 2

Efectúa la función lógica AND entre los argumentos. En otros términos, compara el formato binario de los dos valores y sobre la base de la tabla de la verdad representada en la tabla 1 genera el valor binario apropiado.

Por ejemplo, pulsemos de manera directa:

PRINT BITAND 244 35

intentemos comprender por qué el resultado es

RESULT : 32

Argumento 1	Argumento 2	Resultado
0	0	0
0	1	0
1	0	0
1	1	1

 *Tabla 1.—Función lógica AND.*

Convirtamos el número decimal 244 a binario, hagamos lo mismo con el 35 y coloquémoslos uno debajo del otro como sigue:

Decimal 244=binario 1 1 1 1 0 1 0 0
Decimal 35=binario 0 0 1 0 0 0 1 1

Examinemos el primer bit (por la izquierda) del número 244 (vale 1) y realicemos la operación AND (haciendo referencia a su tabla de la verdad) con el primer bit del número 35 (vale 0).

La línea de la tabla 1 que nos interesa es, por lo tanto, la tercera: las entradas que tenemos son 1 y 0, como salida obtendremos un bit desactivado (vale 0).

Repitamos el mismo razonamiento para los restantes 7 bits de los dos números:

Bit nº	Número 1	Número 2	Resultado
2	1	0	0
3	1	1	1
4	1	0	0
5	0	0	0
6	1	0	0
7	0	1	0
8	0	1	0

La cuarta columna recoge los valores que se derivan de las sucesivas operaciones.

Esta es pues la representación binaria del resultado buscado por nosotros (BITAND 244 35).

Coloquemos de forma horizontal los bits obtenidos en la última columna, añadiendo el del bit número 1:

0 0 1 0 0 0 0 0

Vamos a ver ahora cómo podemos convertir este número binario obtenido en el correspondiente decimal, que, como esperamos, tendrá que ser 32.

El algoritmo a seguir es el siguiente: se vuelve a escribir en orden invertido la secuencia de bits encontrada, obteniendo:

0 0 0 0 0 1 0 0

Cogemos entonces cada bit, procediendo desde la izquierda a la derecha, y lo multiplicamos por potencias crecientes de dos, comenzando por la potencia "cero" ($2^0=1$).

BIT	VALOR	RESULTADO
PRIMERO	0	$0 * (2^0) = 0$
SEGUNDO	0	$0 * (2^1) = 0$
TERCERO	0	$0 * (2^2) = 0$
CUARTO	0	$0 * (2^3) = 0$
QUINTO	0	$0 * (2^4) = 0$
SEXTO	1	$1 * (2^5) = 32$
SEPTIMO	0	$0 * (2^6) = 0$
OCTAVO	0	$0 * (2^7) = 0$
		Suma = 32

Al sumar los resultados de los productos tendremos el valor decimal correspondiente, que es justo igual a 32.

BITOR argumento 1 argumento 2

Realiza la función lógica OR entre los argumentos. Contrariamente a la función AND, la tabla de la verdad que caracteriza el operador "OR" es la expresada en la tabla 2.

Argumento 1	Argumento 2	Resultado
0	0	0
0	1	1
1	0	1
1	1	1

Tabla 2.—Tabla de la verdad de la función lógica OR.

Para verlo prácticamente pulsemos de manera directa:

PRINT BITOR 244 35

(que son los mismos números utilizados en la discusión de la función AND).

La computadora contestará con el mensaje

RESULT : 247

Recordemos que los valores decimales 244 y 35 equivalen a los binarios:

Decimal 244=binario 1 1 1 1 0 1 0 0
Decimal 35=binario 0 0 1 0 0 0 1 1

Efectuando el mismo procedimiento que para el operador AND examinemos el bit más a la izquierda del número 244 (=1) y lo comparamos con el primero del número 35, haciendo en este caso referencia a la tabla de la verdad de la función OR.

La línea a tomar en consideración es otra vez la tercera: como entradas tenemos 1 y 0, y como salida, un bit a nivel 1 (encendido).

Si repetimos el mismo razonamiento para los restantes bits de los dos números obtendremos la tabla que sigue:

Bit nº	Argumento 1	Argumento 2	Resultado
2	1	0	1
3	1	1	1
4	1	0	1
5	0	0	0
6	1	0	1
7	0	1	1
8	0	1	1

La última columna, como pasaba en la función AND, es la expresión binaria del número buscado por nosotros (247=BITOR 244 35).

Si queremos, finalmente, convertir en decimal el valor 11101111 (obtenido después de haber invertido la secuencia de bits presentes en la cuarta columna), escribiremos:

BIT	VALOR	RESULTADO
PRIMERO	1	$1 * (2^0) = 1$
SEGUNDO	1	$1 * (2^1) = 2$
TERCERO	1	$1 * (2^2) = 4$
CUARTO	0	$0 * (2^3) = 0$
QUINTO	1	$1 * (2^4) = 16$
SEXTO	1	$1 * (2^5) = 32$
SEPTIMO	1	$1 * (2^6) = 64$
OCTAVO	1	$1 * (2^7) = 128$
		Suma = 247

BITXOR argumento 1 argumento 2

La primitiva BITXOR genera como salida un valor obtenido al aplicar la función OR exclusiva a los dos argumentos.

La tabla de la verdad relativa a este operador es la de la tabla 3.

Argumento 1	Argumento 2	Resultado
0	0	0
0	1	1
1	0	1
1	1	0

Tabla 3.—Función OR exclusiva (XOR) aplicada a dos argumentos.

En lo que se refiere a las técnicas necesarias para la utilización de este operador valen las mismas consideraciones utilizadas para las funciones AND y OR.

Seguidamente les ofrecemos el listado del programa CONVERSION. Después de escribir de manera directa:

RUN CONVERSION NUMERO1 NUMERO2

El programa imprimirá en la pantalla, en formato binario, los dos números de entrada y, siempre en binario, el resultado de realizar con ellos las operaciones AND, OR y XOR.

El programa, durante la fase de la conversión de decimal a binario sigue el algoritmo de las divisiones sucesivas utilizado por nosotros manualmente antes, empleando la primitiva QUOTIENT que veremos en el siguiente apartado:

```
TO CONVIERTE :NUMERO

  MAKE "CONTADOR 0
  MAKE "BINARIO []
  MAKE "RESTO 0
  DECIM.BIN :NUMERO
  PRINT :BINARIO

END


TO CONVERSION :NUMERO1 :NUMERO2

  NODRAW
  PRINT []
  PRINT [LA REPRESENTACION BINARIA]
  PRINT []
  PRINT [DEL PRIMER NUMERO ES:]
  PRINT []
  CONVIERTE :NUMERO1
  PRINT []
  PRINT [Y LA DEL SEGUNDO NUMERO:]
  PRINT []
  CONVIERTE :NUMERO2
  PRINT []
  PRINT [LA OPERACION "AND" DA COMO
    RESULTADO:]
  PRINT []
  CONVIERTE BITAND :NUMERO1 :NUMERO2
  PRINT []
  PRINT [LA OPERACION "OR" PRODUCE EL
    RESULTADO SIGUIENTE:]
  PRINT []
  CONVIERTE BITOR :NUMERO1 :NUMERO2
```



```
  PRINT []
  PRINT [AL USAR LA "OR EXCLUSIVA" NOS
    ENCONTRAMOS CON:]
  PRINT []
  CONVIERTE BITXOR :NUMERO1 :NUMERO2
  PRINT []

END

TO DECIM.BIN :NUMERO

  MAKE "CONTADOR :CONTADOR+1
  IF CONTADOR = 9 THEN STOP
  MAKE "RESTO :NUMERO-2 QUOTIENT
    :NUMERO 2 ;[VER SIGUIENTE APARTADO
    PARA QUOTIENT]
  MAKE "BINARIO SENTENCE :BINARIO :RESTO
  DECIM.BIN QUOTIENT :NUMERO 2

END
```

Operadores algebraicos

El lenguaje LOGO pone a disposición del programador una serie de instrucciones cuya función se desarrolla todo en el campo matemático.

Además de los acostumbrados operadores algebraicos +, *, -, /, podemos disponer de las primitivas:

INTEGER argumento

El argumento puede ser un número, una variable numérica o una expresión.

Cada vez que se utiliza esta instrucción se genera como salida un valor numérico igual a la parte entera del argumento.

Obtendremos, por ejemplo, que

INTEGER 5.1 → 5
INTEGER 5.9 → 5
INTEGER -7.2 → -7
INTEGER -7.8 → -7

O también

INTEGER 65 → RESULT 65

ROUND argumento

También la primitiva ROUND requiere como entrada un valor numérico (constante, variable o también una expresión).
Si el valor de entrada es entero no será alterado; en caso contrario el argumento se redondea al entero más próximo. De hecho, por ejemplo, resultará que:

ROUND 4.3 → 4
ROUND 4.6 → 5
ROUND -7.8 → -8
ROUND -7.2 → -7

y que

ROUND 5654 → 5654

QUOTIENT elemento1 elemento2

La utilidad de esta primitiva se nota cada vez que es necesario calcular un cociente entre dos elementos (constantes y/o variables).
El resultado es el valor entero del cociente; por lo tanto, no tendrá significado una orden del tipo:

PRINT INTEGER QUOTIENT 30 4

He aquí unos ejemplos prácticos del empleo de la primitiva de que hablamos:

PRINT QUOTIENT 8 4 → 2
PRINT QUOTIENT -14 3 → -4
PRINT QUOTIENT 6(-24) → 0
PRINT QUOTIENT 0 534 → 0

Nótese que si el argumento2 es negativo deberá colocarse entre paréntesis.
Es útil que el programador, durante el desarrollo de un algoritmo que utilice la primitiva QUOTIENT, tenga en cuenta que la segunda entrada (divisor) no puede ser nula y que, en consecuen-

cia, si lo fuera, surgiría un mensaje de error que pasa necesariamente la ejecución al procedimiento que se esté efectuando.

REMAINDER argumento1 argumento2

Al contrario de la función desarrollada por QUOTIENT, la primitiva REMAINDER calcula el resto de la división entre los dos argumentos enteros, que pueden ser constantes y/o variables numéricos; en el caso de que el segundo argumento sea negativo, es necesario colocarlo entre paréntesis.
He aquí unos ejemplos:

REMAINDER 10 2 → RESULT : 0
REMAINDER 14 3 → 2
REMAINDER -5 2 → 1
REMAINDER 4(-8) → 0

SQR argumento

Por último, hablemos del operador SQR: éste da la raíz cuadrada del argumento.
Una advertencia para el programador más inexperto: sólo se puede calcular con SQR el valor de la raíz cuadrada de un número positivo o, a lo sumo, nulo.

Funciones trigonométricas

La finalidad de este apartado es guiar al lector para que repase las principales funciones trigonométricas.
Antes de nada es necesario recordar la definición de circunferencia goniométrica: con este término se habla de una circunferencia con centro en el origen de un sistema de ejes cartesianos ortogonales y de radio arbitrario que convencionalmente se supone unitario (Fig. 1).
La circunferencia goniométrica es dividida por los ejes cartesianos en cuatro CUADRANTES.

Primer cuadrante: Arco AB
Segundo cuadrante: Arco BA'
Tercer cuadrante: Arco A'B'
Cuarto cuadrante: Arco B'A

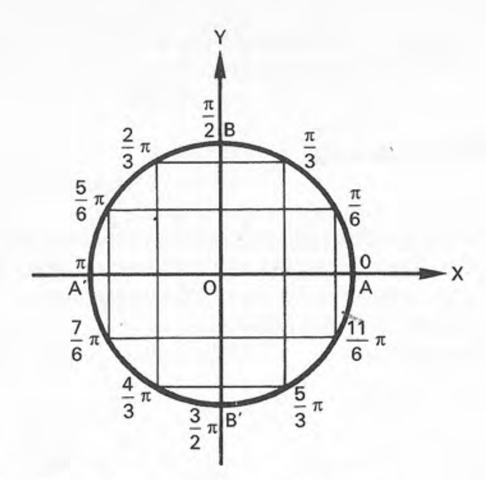


Figura 1.—Circunferencia goniométrica.

Haciendo referencia a esta circunferencia con radio unidad podemos afirmar por definición que:

El seno de un arco es la ordenada del extremo de dicho arco

y que:

El coseno de un arco es la abscisa del extremo de dicho arco

En la figura 2 se puede ver cómo seno y coseno son función de la amplitud del arco y varían según la variación de éste. Para el ángulo elegido (α) el seno sería "Y" y el coseno "X".

Se puede dibujar un diagrama que ponga en relación el valor del ángulo y el seno o coseno correspondiente.

Si sobre las ordenadas llevamos la medida del ángulo y sobre las abscisas el correspondiente valor de la función del seno, obtenemos lo que se representa en la figura 3.

Con el mismo sistema para la función coseno resulta la figura 4.

Estos gráficos resaltan las características más notables de ambas funciones trigonométricas, que pueden ser sintetizadas en:

- ambas son periódicas (con un período de 360º);
- son funciones limitadas; varían, como hemos dicho, desde un mínimo (-1) hasta un valor máximo (+1);
- la función seno es positiva en el primero y en el segundo

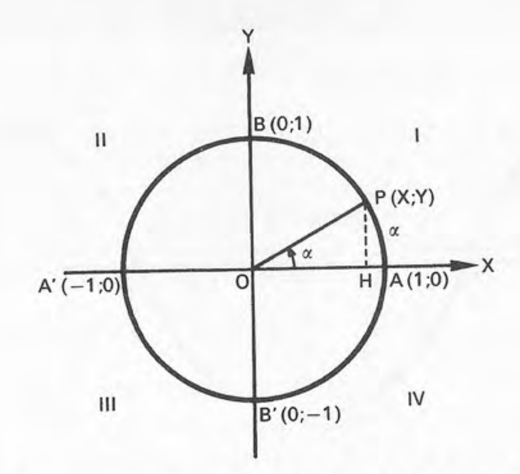


Figura 2.—Visualización gráfica de la relación entre seno, coseno y el arco correspondiente.

cuadrantes y negativa en el tercero y cuarto. Además, la sinusoide es creciente en el primero y cuarto cuadrantes y decreciente en los restantes;

- el coseno es positivo en el primero y cuarto cuadrantes, y negativo en el segundo y tercero. Además, es sencillo verificar en el diagrama que es creciente en los cuadrantes tercero y cuarto, y decreciente en los restantes.

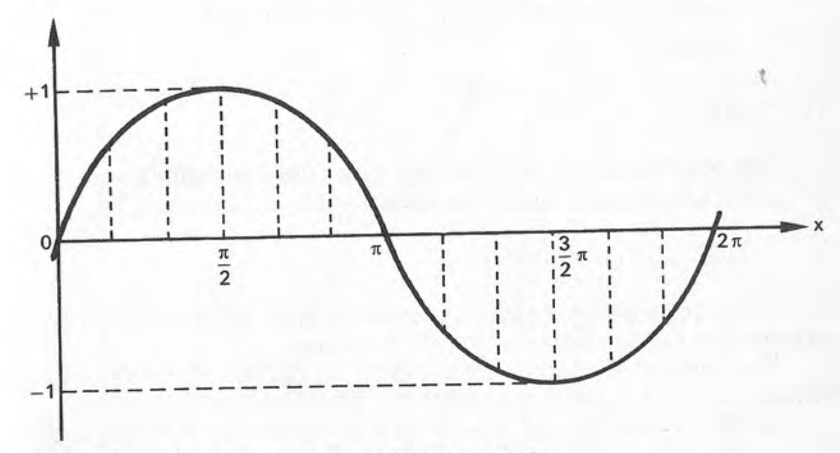


Figura 3.—Diagrama de la función SENO.

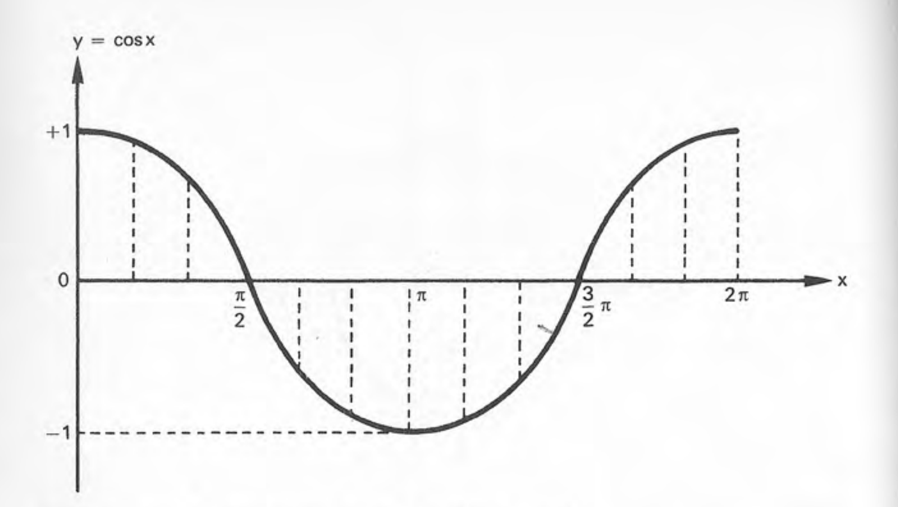


Figura 4.—Diagrama de la función COSENO.

El breve programa siguiente traza en la pantalla el diagrama de la función SENO y el eje de las abscisas (para facilitar una referencia cómoda):

```
TO SENO :X

  IF :X > 360 HOME STOP
  SETXY :X / 2 - 100  100 * SIN :X
  SENO :X + 3

END
```

Restemos atención a la tercera línea del procedimiento: "SIN :X" calcula el seno del argumento,

"SETXY :X/2-100 100*SIN :X"

lleva la tortuga hasta la posición especificada del sistema de ejes cartesianos ortogonales fijados en la pantalla.

Ya que hemos recordado el concepto de seno y coseno, examinemos otra función trigonométrica fundamental: la tangente.

Se llama tangente trigonométrica de un ángulo la relación entre el seno y el coseno del mismo.

En términos matemáticos sería:

TANGENTE (alfa)=SIN (alfa)/COS (alfa)

El diagrama relativo a la función de que hablamos está representado en la figura 5.

La tangente no está definida en los ángulos $\pi/2$, $3\pi/2$, etc. Como verá, son los mismos en los cuales se anula el coseno; esto pasa porque "tang=sin/cos" y, por lo tanto, no puede definirse en los puntos en que hay cero en el denominador.

Un programa sencillo que nos da la posibilidad de trazar el diagrama de la función tangente es:

```
TO GRAFICO :X

  IF :X > 360 HOME STOP
  IF (COS :X) = 0 HOME STOP
  MAKE "Y 100 * (SIN :X) / (COS :X)
  IF :Y < -100 THEN PENUP GRAFICO :X+5
    STOP
  IF :Y > ·100 THEN PENUP GRAFICO :X+5
    STOP
  SETXY :X / 2 - 100 :Y
```

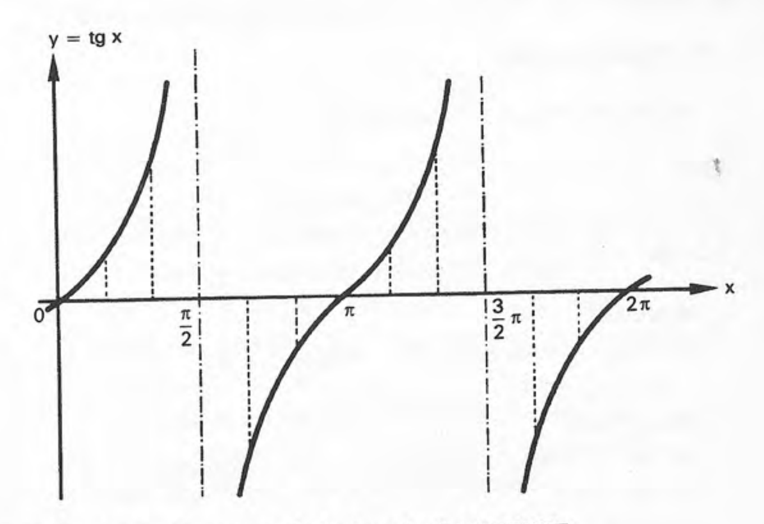


Figura 5.—Diagrama de la función TANGENTE.

```
    PENDOWN
    TANGENTE :X + 3
END

TO TANGENTE
    HOME
    CLEARSCREEN
    GRAFICO 0
END
```

Generación de números aleatorios

El lenguaje LOGO pone a disposición del programador una interesante y, en algunos casos, insustituible función numérica: se trata de la primitiva "RANDOM n", que genera como salida un valor numérico aleatorio, comprendido entre 0 y (n-1).

Es importante recordar que el argumento de esta instrucción debe ser necesariamente positivo y entero.

Un ejemplo de utilización de esta instrucción está constituido por el programa ADIVINE que presentamos a continuación:

```
TO TRANSNUMERO
    OUTPUT FIRST :TENTATIVA
END

TO OK
    NODRAW
    REPEAT 4 [REPEAT 33 [PRINT1 "*]
      PRINT []]
    PRINT []
    PRINT [¡FELICIDADES! LO HA LOGRADO ]
    PRINT []
    PRINT1 SENTENCE [EN SOLO ] :TEN
    PRINT [TENTATIVAS]
```



```
    PRINT []
    REPEAT 4 [REPEAT 33 [PRINT1 "*]
      PRINT []]
    PRINT []
    PRINT []
    PRINT [F = FINAL]
    PRINT [C = CONTINUAR]
    PRINT []
    MAKE "RESPUESTA READCHARACTER
    IF :RESPUESTA = "C THEN ADIVINE
END

TO ENTRADA
    NODRAW
    REPEAT 3 [PRINT []]
    PRINT [INTRODUZCA SU NUMERO]
    PRINT []
    MAKE "TENTATIVA REQUEST
END

TO JUEGO
    MAKE "TEN :TEN + 1
    ENTRADA
    MAKE "TENT TRANSNUMERO
    IF :TENT = :NUMERO.PENSADO THEN OK
    IF :TENT > :NUMERO.PENSADO THEN
      PRINT [] PRINT [MI NUMERO ES MENOR
      QUE EL SUYO] PRINT [] PRINT REQUEST
      JUEGO
    IF :TENT < :NUMERO.PEMSADO THEN
      PRINT [] PRINT [MI NUMERO ES MAYOR
      QUE EL SUYO] PRINT [] PRINT REQUEST
      JUEGO
END
```

```
TO ADIVINE
  MAKE "TEN 0
  NODRAW
  PRINT []
  PRINT [YO PIENSO UN NUMERO]
  PRINT []
  PRINT [COMPRENDIDO ENTRE 0 Y 100]
  PRINT []
  PRINT [USTED DEBERA ADIVINARLO]
  PRINT []
  PRINT [EN EL MENOR NUMERO POSIBLE DE]
  PRINT []
  PRINT [INTENTOS, TENIENDO EN CUENTA]
  PRINT []
  PRINT [TODAS LAS SUGERENCIAS QUE LE]
  PRINT []
  PRINT [IRE HACIENDO]
  PRINT []
  PRINT REQUEST
  MAKE "NUMERO.PENSADO RANDOM 100
  JUEGO
END
```

La finalidad del programa es conseguir en el menor número de intentos la individualización de una cifra generada aleatoriamente por la computadora.

Además, el programa guía al usuario a través de los intentos, advirtiendo si el número propuesto por él es menor o mayor que aquel elegido.

Un ejemplo de ejecución del programa podría ser:

```
RUN ADIVINE

YO PIENSO UN NUMERO
COMPRENDIDO ENTRE 0 Y 100
USTED DEBERA ADIVINARLO
EN EL MENOR NUMERO POSIBLE DE
INTENTOS, TENIENDO EN CUENTA
TODAS LAS SUGERENCIAS
```



```
QUE LE IRE HACIENDO.
<RETURN>
INTRODUZCA SU NUMERO
40
MI NUMERO ES MAYOR
QUE EL SUYO
<RETURN>
INTRODUZCA SU NUMERO
50
MI NUMERO ES MENOR
QUE EL SUYO
<RETURN>
INTRODUZCA SU NUMERO
45
*********************************
*********************************
*********************************
*********************************
¡FELICIDADES! LO HA LOGRADO
EN SOLO 3 TENTATIVAS
*********************************
*********************************
*********************************
*********************************
F=FINAL
C=CONTINUAR
F
```

y el programa termina la ejecución.

Volviendo a la utilización de la primitiva RANDOM es interesante usarla en aplicaciones de tipo gráfico. Aunque aparentemente sencillo, este empleo es, sin embargo, muy difícil. Realizar dibujos agradables al ojo humano quiere decir unir al aspecto "regular" de la imagen unos parámetros aleatoriamente variables: crear un tema de fondo sobre el cual actúa la fantasía de la primitiva RANDOM.

En caso contrario, si confiamos la ejecución de la imagen solamente a lo casual, nos deberemos conformar con verdaderos "líos", como el que se representa en la figura 6, obtenido con el procedimiento DIBUJA.

```
TO DIBUJA
  MAKE "C 0
  ESPIRAL RANDOM 100
```

```
  DIBUJA
END

TO ESPIRAL :X
  FORWARD :X
  RIGHT 90
  MAKE "C :C + 1
  IF :C > 4 THEN STOP
  ESPIRAL :X + 1
END
```

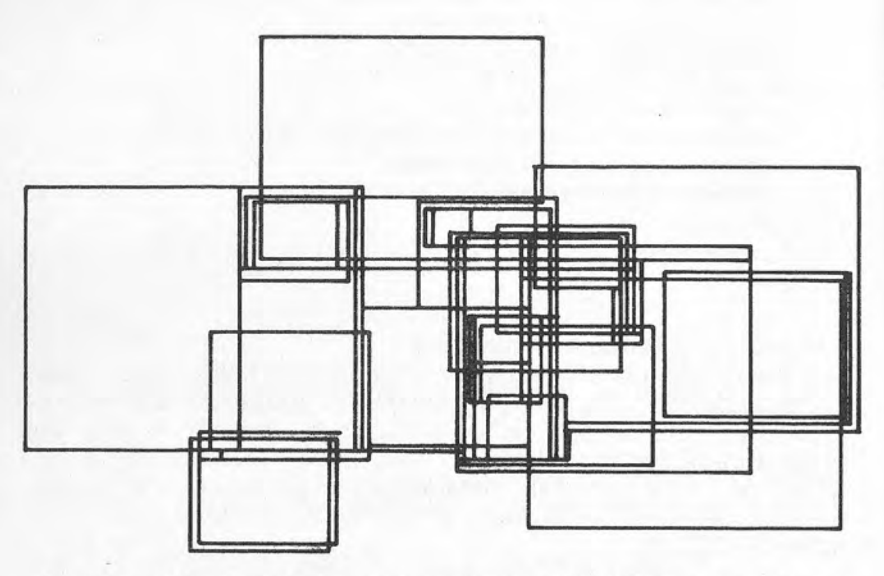


Figura 6.—Resultado del procedimiento DIBUJA, en el que sólo actúa el factor aleatorio.

CAPITULO VI

CIRCULOS, PARABOLAS Y ELIPSES

El lector, sin duda, habrá visto ya que el intérprete LOGO es rico en instrucciones gráficas o no fundamentales cuya función, aunque elemental, es de indiscutible e insustituible valor.

Primitivas como las clásicas FORWARD, LEFT, etc., son muy poco sofisticadas, pero a ellas se encomiendan tareas irrenunciables por parte del mismo lenguaje.

En las páginas anteriores hemos hablado a fondo de cómo poder construir (definir) a través de estas primitivas un conjunto de órdenes más complejas que satisfagan las específicas exigencias del programador.

Este capítulo quiere representar, en este sentido, una ayuda válida para el programador que tenga la intención de enfrentarse con problemas de tipo gráfico, como el dibujo de arcos, círculos o figuras más complejas.

Círculos y arcos

El procedimiento más sencillo que podemos escribir para dibujar una circunferencia es el que sigue:

```
TO CIRCULO

  HOME
  CLEARSCREEN
  PENUP
```

```
    LEFT 90
    FORWARD 100
    PENDOWN
    REPEAT 360 [FORWARD 1 RIGHT 1]

END
```

El procedimiento CIRCULO genera la figura geométrica requerida, sin recurrir a la ecuación matemática típica de la circunferencia, que, además, no es nada fácil; utiliza exclusivamente el principio del "polígono regular".

En otras palabras, crea un polígono de 360 lados tan pequeños que al ojo humano parece una circunferencia.

Este método, desde luego, ofrece sus mejores resultados cuando el lado del polígono tiende a Ø (teóricamente un segmento de longitud nula es igual a un punto).

Podemos decir, por lo tanto, que una circunferencia es aproximable por un polígono regular que tenga como características estos parámetros:

N=número lados (tan grande como sea posible).
L=dimensión de un lado (tan pequeña como sea posible).
LIMITE N*L=perímetro de la circunferencia.
N inversamente proporciona a L de manera que:
N=1/L

En este contexto, y teniendo en cuenta la resolución gráfica de las computadoras, un buen resultado se puede obtener tomando como medidas del lado y del ángulo la unidad.

Obtendremos así el dibujo de un círculo como el de la figura 1.

Figura 1.—Círculo obtenido como un polígono regular de 360 lados.

Si quisiéramos una mejor aproximación sería suficiente cambiar la instrucción REPEAT del programa de esta manera:

REPEAT 720 (FORWARD 0.5 RIGHT 0.5)

Esta manera de trazar una circunferencia es, sin duda, la más sencilla, pero no permite control alguno por parte del usuario en las dimensiones y características de la misma.

Intentemos ahora crear un procedimiento que pueda llevar a cabo una serie de círculos con distintos diámetros.

Pulsemos:

```
TO CIRCULO :LADO

    HOME
    CLEARSCREEN
    PENUP
    LEFT 90
    FORWARD 100
    PENDOWN
    REPEAT 360 [FORWARD :LADO RIGHT 1]

END
```

De esta manera podemos variar las características de la circunferencia dibujada introduciendo un parámetro (lado del polígono) variable.

Pero esta solución resuelve nuestros problemas sólo en parte: si quisiéramos dibujar sencillamente un arco de circunferencia con un cierto ángulo respecto al centro no sería posible.

El programa que sigue permite estas operaciones además de las anteriores.

```
TO CIRCUNFERENCIA :ARCO
    :VARIACION.ANGULAR

    REPEAT :ARCO / :VARIACION.ANGULAR
      [FORWARD  :VARIACION.ANGULAR  RIGHT
      :VARIACION.ANGULAR]

END
```

Intentemos examinar el funcionamiento de este programa, que a primera vista puede aparecer sencillo porque es corto, y que, sin embargo, tiene sus dificultades.

Antes de nada se proporcionan como entradas los parámetros relativos a la medida del arco (en grados) y al valor de la variación angular deseada entre un lado y su adyacente del polígono regular que genera el círculo (si crece este parámetro aumentarán las dimensiones del diámetro).

La instrucción REPEAT tiene como primer argumento (número de ciclos) el resultado de la división entre ARCO y VARIACION.ANGULAR que queda fijada antes de que varíe este último parámetro.

En otras palabras, si introducimos estos valores

ARCO: 100 grados
VARIACION.ANGULAR : 2 grados

La repetición determinada por (ARCO/VARIACION.ANGULAR) será de 50.

Para obtener una circunferencia (recordar que no es, en definitiva, nada más que un arco de 360 grados) deberíamos pulsar

RUN CIRCUNFERENCIA 360 1

Estudiemos ahora, antes de examinar aspectos más complejos de las curvas, algunos programas demostrativos.

CIRCUNFERENCIAS TANGENTES

```
TO CIRC.TANG

  HOME
  CLEARSCREEN
  PENUP
  LEFT 90
  FORWARD 100
  PENDOWN
  RIGHT 90
  REPEAT 360 [FORWARD 1 RIGHT 1]
  REPEAT 360 [FORWARD 1.5 RIGHT 1]
  REPEAT 240 [FORWARD 1 RIGHT 1.5]

END
```

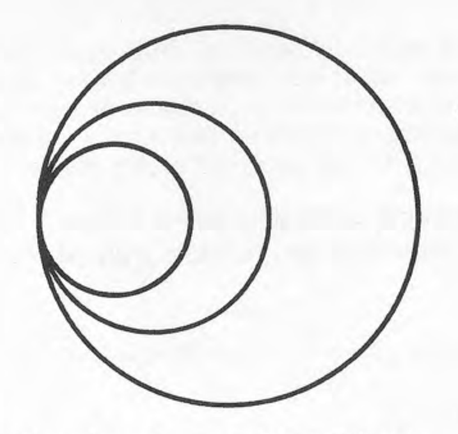



Figura 2.—Circunferencias tangentes interiores.

Genera la figura 2.

CIRCUNFERENCIAS EXTERIORES

```
TO CIRC.EXT

  REPEAT 360 [FORWARD 1 RIGHT 1]
  REPEAT 360 [FORWARD 1 LEFT 1]

END
```

Dibuja la figura 3.

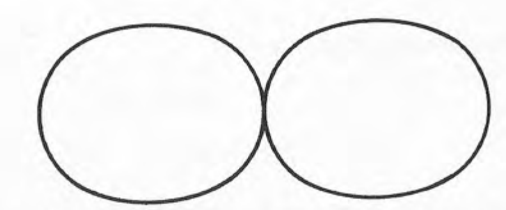


Figura 3.—Dos circunferencias tangentes exteriores.

```
TO EJES

   HOME
   CLEARSCREEN
   SETXY 155 0
   HOME
   SETXY -155 0

END
```

Al realizar

RUN PARABOLA 20

obtendremos lo que muestra la figura 5.

Nótese cómo al incrementarse el valor del parámetro "A" disminuye el crecimiento hiperbólico de los lados de la parábola. En otros términos, un número pequeño significa una figura más "cerrada", y viceversa.

Elipses

Matemáticamente la elipse es el lugar geométrico de los puntos del plano cuyas distancias a dos puntos fijados (f1 y f2) llamados focos tienen una suma constante.

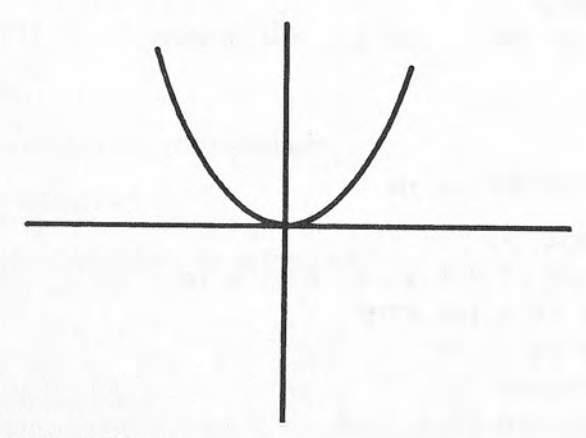


Figura 5.—Parábola.

La ecuación general es:

$(Y^2/A^2)-(X^2/B^2)=1$

Para dibujarla usaremos el siguiente procedimiento:

```
TO ELIPSE :A :B

   EJES
   PENUP
   HIDETURTLE
   PLOTTER - :A :A :B 1

END


TO PLOTTER  :X :A :B :INC

   IF (:X * :X) > (:A * :A) STOP
   IF :X = :A THEN MAKE "INC (- :INC)
   SETXY :X :INC * :B * SQR (1 - (:X *
     :X) / (:A * :A))
   PENDOWN
   PLOTTER :X + :INC :A :B :INC

END


TO EJES

   HOME
   CLEARSCREEN
   SETXY 155 0
   HOME
   SETXY -155 0

END
```

Para ejecutar lo que hemos escrito pulse:

RUN ELIPSE 100 180 10 10

que nos dará el resultado representado en la figura 6.

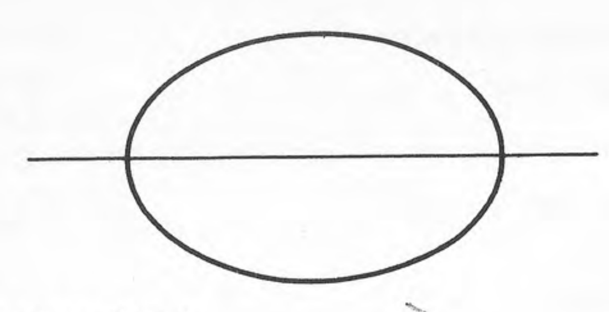


Figura 6.—Elipse.

Tenga en cuenta que el primer parámetro corresponde a la medida del eje mayor y que el segundo se refiere a la medida del eje menor.

De esta manera se puede obtener una elipse con el eje mayor paralelo al de las abscisas:

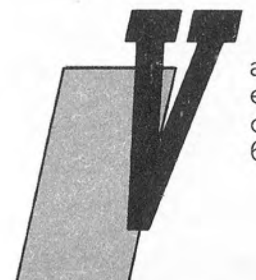
RUN 100 100 10 10

o paralelo al de las ordenadas:

RUN 30 100 10 10

CAPITULO VII

LOS FICHEROS



Vamos a ver aquí los comandos dedicados en el LOGO al tratamiento de los ficheros, típicos de las versiones basadas en el COMMODORE 64 y en el Apple II.

Versión COMMODORE 64

SAVE "nombrefichero

Traslada todos los procedimientos presentes en memoria al disco, dando al fichero el nombre especificado.

SAVE "nombrefichero [Proce1 Proce2...]

Abre sobre el disco un fichero con el nombre especificado y traslada a él exclusivamente los procedimientos especificados como argumentos, a condición que estén presentes en memoria.

READ "nombrefichero

Carga del disco y del fichero deseado las primitivas contenidas que, si son distintas a las presentes en la memoria, se añaden a ellas; en caso contrario las sustituyen.

Una técnica particularmente interesante puede ser hacer que, después de una instrucción READ, el programa cargado en memoria sea ejecutado automáticamente.

Supongamos tener residente en memoria el programa PRG que deseamos registrar sobre disco con la técnica de ejecución automática (AUTOSTART).

Pulsamos el comando:

MAKE "STARTUR [PRG]

que crea y coloca al fondo del listado de nuestro programa la variable STARTUP que contiene la lista [PRG].

En este momento grabemos sobre disco el programa PRG:

SAVE "PRG

y la operación estará concluida.

Si ahora queremos cargar nuestro programa (mejor después de un ERASE ALL) es suficiente escribir:

READ "PRG

Después de haber llevado a cabo la carga será ejecutado automáticamente sin necesidad de nuestra intervención.

BSAVE "nombrefichero star end+1

Esta primitiva salva en disco una región entera de memoria; más exactamente desde la dirección "star" hasta la "end" dándole el nombre "nombrefichero.LOGO". Las direcciones "star" y "end" deben ser dos valores decimales enteros y positivos.

BLOAD "nombrefichero

Ejecuta la función contraria de BSAVE. Coloca en memoria el contenido del fichero indicado, generado anteriormente gracias al mando BSAVE.

SAVEPICT "nombrefichero

Copia en el fichero de nombre especificado el contenido de la página de alta resolución.

La opción SAVEPICT crea sobre disco dos ficheros: el primero contiene las informaciones gráficas, el segundo los colores.

READPICT "nombrefichero

Si anteriormente hemos registrado una figura con el mando SAVEPICT podemos visualizarla de nuevo utilizando esta primitiva. Evidentemente sólo tiene sentido en el caso de que el fichero buscado exista; en caso contrario, el sistema genera un mensaje de error.

CATALOG

Visualiza la lista de los nombres de todos los ficheros presentes en el disco introducido en la unidad.

ERASEFILE "nombrefichero

Borra del disco el fichero especificado en el argumento. Si no existe fichero con el nombre buscado genera un mensaje de error.

DOS [NEW:nombredisco, ID]

Ejecuta la operación de formateo de disco necesaria para preparar un disco virgen, nunca utilizado, para las futuras grabaciones. La identificación (ID) es un número entero de dos cifras que se adjudica al disco junto con el nombre, para facilitar sus futuras búsquedas.

DOS [RENAME:nuevonombre=viejonombre]

Supongamos haber registrado sobre disco un fichero llamado "ANTES" y que queremos cambiar su nombre por el de "DESPUES"; será suficiente utilizar la primitiva RENAME pulsando

DOS [RENAME:DESPUES=ANTES]

La operación es automática.

DOS [COPY:nombrecopia=ficheroacopiar]

La primitiva COPY abre el fichero llamado "nombrecopia" y copia en él el contenido del fichero "ficheroacopiar".

La operación es posible si en el disco está presente el fichero a duplicar y si el nombre del fichero duplicado no corresponde a ningún fichero ya presente en el disco.

DOS [SCRATCH:nombrefichero]

Esta instrucción es análoga a la primitiva ERASEFILE; borra sin remedio del disco el fichero cuyo nombre está especificado en el argumento.

Determina como salida un listado con sólo dos elementos: el primero es el valor actual de la abscisa del cursor, el segundo constituye la ordenada de éste.

DOUBLECOLOR X

Dispone la pantalla gráfica de modo doble color; la resolución horizontal disminuye (150 puntos) pero se pueden utilizar hasta 16 colores diferentes (y también dos simultáneamente en la misma región de 8×8 puntos) en base a la tabla 1 del primer capítulo.

Tenga en cuenta que si se trabaja en manera de doble color la instrucción STAMPCHAR provoca resultados anómalos (caracteres ilegibles) por el cambio de la resolución horizontal (y por factores más complejos que ahora no es oportuno tratar).

DRAWSTATE

Potente comando que genera un listado que contiene todas las informaciones actuales de la tortuga en el siguiente orden:

- estado de la tortuga (FALSE=Plumilla levantada/TRUE=Plumilla bajada);
- visibilidad de la tortuga (FALSE=tortuga no visible/TRUE=tortuga visible);
- código color pantalla (número entero desde 0 a 15);
- color de la tortuga;
- modo color (estándar o doble color, o bien singlecolor o doublecolor);
- modo visualización (gráfica, texto o gráfica y texto);
- código del color atribuido a la pantalla en baja resolución;
- color de los caracteres.

El listado generado por DRAWSTATE después del encendido de la máquina es: [TRUE TRUE 11 1 DRAW SINGLECOLOR SPLITSCREEN 14 1]

FPRINT X

En el caso de que el argumento "X" sea una tabla se imprime en la pantalla con paréntesis, contrariamente a lo que pasa para la clásica y más utilizada PRINT (o PRINT1).

HEADING

Genera el valor en grados del ángulo tomado por la tortuga. Recordamos que después de instrucciones del tipo HOME, CS y DRAW la instrucción HEADING genera el valor 0 (cero grados).

JOYSTICK X

La variable "X" debe asumir exclusivamente los valores 0 ó 1; la primitiva determina la actual posición del joystick introducido en la puerta número "x".

JOYBUTTON X

Como JOYSTICK requiere que el parámetro "x" valga 0 ó 1; el comando JOYBUTTON genera el mensaje TRUE si el botón del joystick se pulsa; en caso contrario tendremos la señal FALSE.

NOPRINTER

Después de una operación sobre impresión empezada gracias al mando PRINTER, NOPRINTER cierra los canales de comunicación anteriormente abiertos.

NOWRAP

En el modo gráfico estándar (WRAP), en el caso de que la tortuga salga por un lado de la pantalla, volverá a entrar inmediatamente por el lado contrario. Es posible gracias a la instrucción NOWRAP evitar esto. Si intentamos escribir

HOME NOWRAP FORWARD 5000

obtenemos de hecho el mensaje de error "TURTLE OUT OF BOUND" (muy cerca de la expresión española "tortuga fuera de la pantalla"). Para volver al modo gráfico normal es suficiente recurrir a la expresión WRAP.

PADDLE X

Es posible controlar la actual posición del Paddle número "x" gracias a esta interesante primitiva.

PADDLEBUTTON x

Genera el mensaje TRUE o FALSE, dependiendo de que la tecla del Paddle número "x" se pulse o no.

PRINTER

Al ejecutarlo, las acostumbradas primitivas de impresión sobre pantalla actuarán directamente sobre la impresora. Para volver al modo estándar basta pulsar NOPRINTER.

SETH X, SETHEADING X

Fija el ángulo de la tortuga en el valor especificado por el argumento "X".

SETSHAPE X

El parámetro "x" debe de ser de tipo numérico e incluido entre 0 y 7 (entero); este comando asigna a la plumilla activa (en total son 8) la imagen del "sprite" (duende, figura) número "x" (estos también son 8). En el momento del encendido a la plumilla número 0 corresponde el sprite 0, a la número 2 el sprite 2, y así sucesivamente.

SHAPE

Da el valor del argumento de la última instrucción SETSHAPE que ha sido efectuado; en el caso de que pulsemos SHAPE sin haber llevado a cabo anteriormente la primitiva SETSHAPE, obtendremos el valor 0.

SINGLECOLOR

Lleva el modo de visualización de alta resolución al formato estándar si anteriormente había sido alterado por la primitiva DOUBLECOLOR.

TELL x

El valor de la variable "x" debe ser entero e incluido entre 0 y 7. Probablemente no todos saben que el LOGO basado en sistemas COMMODORE pone a disposición del programador 8 sprites. Es posible usarlos todos, pero sólo pueden moverse de uno en uno. Por ejemplo, escribiendo TELL 3, todas las instrucciones que pulsemos después serán asignadas automáticamente a la tercera plumilla. En el momento del encendido la máquina trabaja con el SPRITE número 0 habiendo dispuesto el estado de los restantes 7 (véase DRAWSTATE) al valor (TRUE, TRUE...); en otras palabras, si deseamos dibujar por medio de la plumilla número "x" (con x≠0) hace falta ante todo pulsar la secuencia TELL x PENDOWN (baja la plumilla "x") SHOWTURTLE (haz visible la tortuga).

TOWARDS x y

Potente primitiva gráfica. Genera el valor del ángulo que la tortuga tomaría desplazándose desde la posición en que se encuentra hasta el punto de coordenadas "x" e "y". Si escribimos, por ejemplo, HOME INTEGER TOWARD 100 100 obtendremos el valor entero 45, ya que la recta que une el punto (0, 0), donde se encontrará la tortuga después de DRAW, con el punto (100, 100), localizado por los argumentos de la primitiva TOWARDS, está inclinada justo 45 grados con respecto a los ejes cartesianos.

WHO

Devuelve un valor numérico entero e incluido entre 0 y 7 que corresponde al número de la plumilla con que actualmente estamos trabajando. Véase a este propósito el comentario relativo a la primitiva TELL x. Como en el momento del encendido el sistema trabaja con la plumilla número 0, pulsando WHO obtendremos por lo tanto 0.

WRAP

Ver NOWRAP.

XCOR

Devuelve el valor entero actual de la abscisa de la tortuga.

YCOR

Devuelve el valor entero actual de la ordenada de la tortuga.

.ASPECT x

Potente instrucción con posibilidad de cambiar la escala de la definición gráfica horizontal. Muy a menudo ocurre que se trabaja con monitores que tienen como características una pantalla más ancha que alta: la imagen que resulta está claramente distorsionada y molesta además de ser estéticamente imprecisa (círculos que parecen elipses o cuadrados que se vuelven rectángulos). Para evitar este inconveniente el intérprete LOGO pone a disposición del usuario esta primitiva: si pulsamos ".ASPECT x" podremos (a condición de que "x" sea distinto del valor estándar 0,76) AUMENTAR o DISMINUIR la escala resolutiva vertical.

CALL x

Pasa todo el control del sistema a una subrutina en lenguaje máquina residente en la memoria a partir de la dirección decimal "x" especificada en el argumento; el usuario que tenga capaci-

dad de llevar a cabo programas en lenguaje máquina sabe que para volver al ambiente del intérprete LOGO es suficiente una insstrucción del tipo RTS.

.CONTENTS

Lista los nombres de todos los procedimientos y de las variables actualmente residentes en la memoria de la computadora.

.DEPOSIT x y

Los argumentos de esta primitiva deben de ser ambos enteros y positivos (a lo sumo nulos); en particular, la variable "x" debe de estar incluida entre un mínimo de 0 y un máximo de 256*256, mientras el valor "y" debe ser necesariamente menor de 256. La instrucción ".DEPOSIT x y" copia el contenido del argumento "x" en la posición "y" de la memoria RAM.

.EXAMINE x

Genera el valor entero, incluido entre 0 y 255, leído en la posición de memoria ROM o RAM número "y".

.GCOLL

El intérprete necesita periódicamente mejorar la disposición de las variables de memoria; esta operación, llamada garbage collection, es llevada a cabo automáticamente por el sistema; sin embargo, se deja al programador la posibilidad de utilizarla si es necesaria.

.NODES

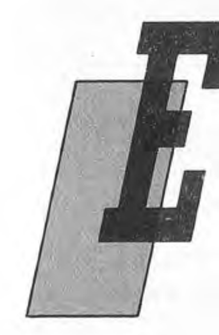
Interesante comando cuya función es poner en conocimiento del programador cuánta memoria RAM dispone todavía para el desarrollo y la ejecución de sus programas. Se aconseja utilizar la instrucción .NODES haciéndola preceder por la primitiva .GCOLL.

.SPRINT x

La primitiva SPRITE-PRINT es, sin duda, de gran ayuda práctica para el programador que tenga la intención de adquirir una preparación completa acerca de la manipulación de los sprites. Pulsando ".SPRINT x" el intérprete LOGO traslada a la pantalla de baja resolución lo que está codificado en la memoria de definición relativa al sprite número "x".

CAPITULO IX

APPLE II Y LOGO



En este capítulo se examinan las principales primitivas del lenguaje con referencia a la versión basada en el APPLE II.

Una buena parte de estas instrucciones han sido ampliamente discutidas; en este caso, al lado de cada uno se nombra el capítulo en el cual se ha hablado de ellas.

Primitivas gráficas

BG, BACKGROUND X

Llena la página gráfica con el color correspondiente al código "x" propuesto.

CHAR X

Genera el carácter cuyo valor ASCII es "x"; el argumento de esta primitiva, por lo tanto, debe de ser positivo (a lo sumo nulo), entero y menor de 255.

CLEAN

Borra el contenido de la página gráfica sin cambiar los parámetros (posición, ángulo, etc.) de la tortuga.

CS, CLEARSCREEN (Capítulo 1)

CLEARTEXT

Como CLEARSCREEN, pero actúa en la pantalla de baja resolución gráfica.

CURSOR

Da como salida la posición actual del cursor.

DOT x

Enciende un puntito en la pantalla en correspondencia con el valor de "x".

FD, FORWARD x (Capítulo 1).

FULLSCREEN (Capítulo 1).

HEADING

Devuelve el ángulo tomado por la tortuga.

HT, HIDETURTLE (Capítulo 1).

HOME (Capítulo 1).

PEN

Genera un listado que contiene el tipo y color actualmente característicos de la plumilla.

PC, PENCOLOR (Capítulo 1).

La tabla de códigos- colores típica del sistema APPLE II es la que sigue:

CODIGO	COLOR
00	Negro
01	Blanco
02	Gris
03	Violeta
04	Naranja
05	Azul marino

PD, PENDOWN (Capítulo 1).

PE, PENERASE (Capítulo 1).

PX, PENREVERSE

Pone en vídeo inverso el recorrido de la tortuga.

PU, PENUP (Capítulo 1).

SETH X, SETHEADING X

Coloca la plumilla dándole un ángulo de "x" grados; las coordenadas, sin embargo, no varían.

SETPC n

Da a la plumilla el color del código "x".

SETPEN X

X debe de ser una lista que contenga tipo y color de la plumilla. La primitiva se encargará de la asignación de estos parámetros a la tortuga.

SETPOS X

Mueve la tortuga hasta la posición definida por el parámetro "x".

SETX x (Capítulo 1).

SETY y (Capítulo 1).

SHOWNP

Comprueba si la tortuga es visible; en tal caso genera el mensaje TRUE; si es al contrario, la salida es FALSE.

ST, SHOWTURTLE (Capítulo 1).

SPLITSCREEN (Capítulo 1).

TEXTSCREEN (Capítulo 1).

Dispone la pantalla en modo texto.

WINDOW

Crea el llamado "modo ventana". La tortuga tiene la posibilidad de moverse en un espacio más amplio del representado por la pantalla.

WRAP

Después de esta orden, al salir la tortuga por un determinado lado de la pantalla vuelve a entrar por el lado contrario.

XCOR

Devuelve la abscisa de la tortuga.

Devuelve la ordenada de la tortuga.

Otras primitivas

AND x y (como BITAND, Capítulo 5).

ARCTAN x

Calcula qué ángulo tiene una tangente cuyo valor está especificado por el argumento "x".

ASCII x (Apéndice B).

BURY pkg

Borra todos los procedimientos contenidos en pkg.

BF X, BUTFIRST X (Capítulo 3).

BL X, BUTLAST X (Capítulo 3).

BUTTO x

Controla si la tecla del paddle número "x" está pulsada; en caso afirmativo genera un mensaje TRUE, al contrario tendremos FALSE.

CATCH nombrelista

Ejecuta la lista de instrucciones.

CO

Vuelve a coger la ejecución del procedimiento momentáneamente suspendido por el comando PAUSE.

COPYDEF X Y

Potente primitiva que permite el inmediato duplicado de los contenidos del procedimiento "x" en el procedimiento "y".

COS x (Capítulo 5).

COUNT x

Si "x" es una lista que contiene "n" elementos, al aplicarle esta orden nos devolverá el valor "n".

EDNS x

Introduce el operador de modo EDITOR pero sólo para las variables contenidas por el procedimiento "x"; es conveniente proporcionar como argumentos de EDNS dos o más nombres de procedimientos.

EQUALP x y (como x=y).

ERALL (como ERASE ALL, Capítulo 2).

ERASE x (Capítulo 2).

ERN X

Borra de la memoria la variable "x". Está permitida una orden del tipo ERNS x y z; las tres (o más) variables serán anuladas en orden.

ERNS x

El parámetro "x" debe ser el nombre de un procedimiento actualmente definido; todas las variables contenidas en él se cancelan. Están permitidas varias entradas (por ejemplo: ERNS x y z).

FIRST x (Capítulo 3).

FPUT x y (Capítulo 3).

GO x

Efectúa un salto incondicional de la ejecución del programa hasta la dirección (label) llamada "x". El programador capaz NO debería utilizar esta primitiva.

IF (Capítulo 4)

IFF, IFFALSE (Capítulo 4).

IFT, IFTRUE (Capítulo 4).

INT n (Capítulo 5).

ITEM x

Si el argumento "x" está compuesto por "y" elementos (y>x) se genera entonces como salida el elemento que ocupa la posición "x" especificada por el argumento.

LAST (Capítulo 3).

LT x, LEFT x (Capítulo 1).

#LIST x y (como LIST, Capítulo 3).

#LOCAL x (como LOCAL, Capítulo 3).

LPUT x y (Capítulo 3).

MAKE x y (Capítulo 3).

MEMBER x y

Se genera el mensaje TRUE si el elemento "x" está comprendido en la lista "y"; en caso contrario resulta FALSE.

NOT Predicado

Genera una decisión de tipo TRUE si el predicado es FALSE, y viceversa.

NUMBER x

Genera una decisión de tipo TRUE si el objeto "x" es un número.

#OR (como BITOR, Capítulo 5).

OP, OUTPUT (Capítulo 3).

PAUSE

Introduce una pausa en la ejecución del procedimiento.

PO x (Capítulo 2).

POALL (como PO ALL, Capítulo 2).

POS

Devuelve la actual posición de la tortuga.

POTS x (Capítulo 2).

PRIMITIVEP x

En el caso de que el argumento "x" corresponda al nombre de una primitiva del lenguaje se genera el mensaje TRUE (FALSE en el caso contrario).

#PRINT x (como PRINT, Capítulo 3).

#PRODUCT x y

Calcula el producto de los argumentos.

QUOTIENT x y

Efectúa la división (x/y) y trunca su resultado.

RANDOM n (Capítulo 5).

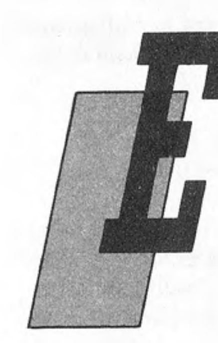
RC, READCHARACTER (Capítulo 3).

REMAINDER X Y

Devuelve el resto de la división entre los argumentos ("x" dividido por "y").

APENDICE A

MENSAJES DE ERROR



En este apéndice se examinan los mensajes con los cuales nos avisa el LOGO en el caso de que el programa presente errores sintácticos y/o lógicos.

El análisis se desarrolla sobre los sistemas Commodore 64 y Apple II.

Commodore 64

(01): (Procedimiento) didn't output

Este mensaje aparece cuando un procedimiento o una primitiva que tienen necesidad de argumentos están llamados con otros procedimientos o primitivas en lugar de los valores numéricos requeridos.

Ejemplo:

INSTRUCCION: CUADRADO FORWARD 100
error: CUADRADO didn't output

(02): (Primitiva) doesn't like (datos) as input

Indica una operación entre dos tipos de datos incompatibles, por ejemplo, una multiplicación entre un número y una lista.

(03): (Primitiva) doesn't like (datos) as input it expects true or false

Aparece en las instrucciones if... then... else... not, allof, anyof, en que la computadora espera una proposición verdadera o falsa y se encuentra en la situación de deber evaluar una expresión del tipo:

if: x - 10 then print [final]

que no es ni verdadera ni falsa; la contestación será:

if doesn't like 2 as input. it expects true or false

(04): (mensaje) in line
(línea) al level (nivel) of (nombre procedimiento)

Es éste un tipo de mensaje que el intérprete va construyendo; expone el nivel en que se ha parado el programa, la línea equivocada y el tipo de error.
El error que sigue es un ejemplo de esto.

(05): there is no procedure named (procedimiento), in line (línea de Programa) at level (x) of (procedimiento).

Ha sido llamado un procedimiento inexistente en la línea de programa listado, a nivel de operaciones (x) del procedimiento llamado. Por ejemplo:

There is no procedure named fd 100, in line fd 100
at level of cuadrado

(06): (nombre) is a logo primitive.

El LOGO advierte que no es posible definir nuestro procedimiento con el nombre elegido, ya que representa una primitiva del lenguaje.

(07): (Procedimiento) needs more inputs.

El procedimiento llamado tiene necesidad de más datos.

(08): (Primitiva) needs more inputs.

La primitiva llamada tiene necesidad de más datos.

(09): (Operador aritmético) needs something before it.

Se ha olvidado un operador aritmético en un cálculo numérico.

Ejemplo:

Print/21

Contestación: "/needs something before it".

Es decir, falta el número que hay que dividir por 21.

(10): (Primitiva) should be used only inside a Procedure.

Advierte que la primitiva empleada por nosotros no puede ser utilizada de manera directa sino sólo en un procedimiento.

Ejemplo:

output should be used only inside a Procedure

(11): can't divide by cero

El mensaje 11 indica que se está llevando a cabo una división por cero (absurdo desde el punto de vista matemático) que el ordenador no puede efectuar.

(12): disk error

Error debido a la unidad de disco.

(13): end should be used only in the editor

Indica el uso equivocado de la primitiva, "END", como por ejemplo: "Print 8 end" pulsando de manera directa.

(14): else is out of place

Con este mensaje se señala la presencia de un "else" en el interior de una instrucción sin "if".
Recordemos que, de hecho, no es posible escribir un else fuera de instrucciones "if... else".

(15): file not found

El fichero requerido no ha sido encontrado; se aconseja examinar el directorio (catálogo) para estar seguros de la presencia del fichero.

(16): line given to define too long

El argumento de "define" es superior a 256 caracteres y, por lo tanto, es rechazado por la computadora.

(17): line given to repeat too long

Lo mismo del error anterior con "repeat" en lugar de "define".

(18): line given to run too long

Como el número 16 con "run" en lugar de "define".

(19): no storage left!

Hemos agotado la memoria a nuestra disposición. La computadora nos invita a no intentar memorizar nuevas cosas antes de haber borrado algún procedimiento con ERASE.

(20): number too large

El argumento numérico de la operación es demasiado grande.

(21): number too small

El argumento numérico de la operación es demasiado pequeño.

(22): procedure nesting too deep

Han sido introducidos más de 200 procedimientos, número máximo aceptado por el Commodore 64.

(23): the disk is full

El disco en que estamos registrando los programas está completo.

(24): the disk is write protected

El disco está protegido de la escritura; no se grabó nada.

(25): there is no disk in the disk drive

El disco no está presente en su unidad.

(26): turtle out of bounds

La tortuga ha pasado los límites de altura y/o largo de la pantalla.

Apple II

(01): (Procedimiento) is already define

El procedimiento ya ha sido definido.

(02): number too big

El valor numérico es excesivamente grande.

(03): (Símbolo) isn't a procedure

No es una palabra del vocabulario LOGO, es decir, no existe este procedimiento.

(04): (Símbolo) isn't a word

Lo especificado entre paréntesis no es una primitiva.

(05): (Procedure) can't be use in a procedure

No ha sido utilizado correctamente el procedimiento nombrado entre paréntesis.

(06): (Símbolo) is a Primitive

Este mensaje aparece cuando se intenta definir un procedimiento con el nombre de una primitiva.

(07): (Símbolo) is undefined

Lo que estamos buscando no ha sido definido.

(09): I'm having trouble with the disk

Hay problemas por lo que se refiere a la lectura del disco.

(10): disk full

El disco está completo; es necesario introducir otro o abrir espacio en el mismo.

(11): Can't divide hoy cero

Se está efectuando una división por cero o una operación matemáticamente imposible.

(12): end of data

Han terminado los datos.

(13): file alreading exists

El fichero que queremos crear (en disco) ya existe.

(14): file locked

Señala que el fichero se encuentra ya "grabado" en disco.

(15): file not found

El fichero requerido no ha sido encontrado, no está en el disco introducido o es necesario poner "on line" la unidad de disco.

(16): file is wrong type

Ha sido requerido (o introducido) un fichero de tipo equivocado.

(17): no more file buffers

No es posible añadir más instrucciones. El buffer está al completo.

(18): (Símbolo) not found

No ha sido encontrado lo que se ha pedido.

(19): (Símbolo) is not true or false

La primitiva no es verdadera ni falsa. Aparece en las propuestas de "if... then... else" y de manera particular, en "anyof" y "allof".

(20): Pausing...

Señal de espera.

(21): stopped

Hemos interrumpido un procedimiento en marcha pulsando CTRL+9.

(22): not enough inputs to (Procedimiento)

No son suficientes las entradas proporcionadas al procedimiento en marcha.

(23): too many inputs to (Procedure)

Lo contrario del anterior.

(24): too much inside parentheses

Demasiadas instrucciones están encerradas entre paréntesis.

(25): can only do that in a Procedure

Es posible operar de tal manera sólo en el interior de un procedimiento.

(26): turtle out of bounds

La tortuga ha pasado los límites de la página gráfica.

(27): don't know how to (símbolo)

La computadora no reconoce esta instrucción como un procedimiento ni como una primitiva: controlar la sintaxis.

(28): (Símbolo) has no value

El "símbolo" no tiene un valor definido.

(29): don't know what to do with (símbolo)

La computadora no sabe cómo ejecutar esta instrucción: controlar la sintaxis.

(30): disk is write protected

El disco está protegido de la escritura. No se grabó nada.

(31): (Procedure) doesn't like (símbolo) as input

Este procedimiento no es admisible como entrada.

APENDICE B

EL CODIGO ASCII

En este apéndice se muestran las tablas relativas a los códigos ASCII del COMMODORE y del APPLE II.

Los comandos que permiten la utilización de los caracteres ASCII son:

CHAR

Genera el valor numérico correspondiente al carácter deseado; su sintaxis es:

CHAR carácter

ASCII

Devuelve el carácter correspondiente al valor propuesto según la sintaxis:

ASCII número

En la primera columna de las tablas se da el valor numérico decimal, y en la segunda, el carácter ASCII correspondiente.

COMMODORE

CODIGO	CARACTER	CODIGO	CARACTER
ØØ		26	
Ø1		27	
Ø2		28	RED
Ø3		29	CRSR →
Ø4		3Ø	GRN
Ø5	WHY	31	BLU
Ø6		32	SPACE
Ø7		33	!
Ø8	desactiva SHIFT C=	34	"
Ø9	activa SHIFT C=	35	#
1Ø		36	$
11		37	%
12		38	&
13	RETURN	39	'
14	VIDEO INVERSO	4Ø	(
15		41	)
16	CRSR ↓	42	*
17	RVS OFF	43	+
18	CLR HOME	44	,
19	INST DEL	45	–
2Ø		46	.
21		47	/
22		48	0
23		49	1
24		5Ø	2
25		51	3

COMMODORE (Continuación)

CODIGO	CARACTER	CODIGO	CARACTER
52	4	78	N
53	5	79	O
54	6	8Ø	P
55	7	81	Q
56	8	82	R
57	9	83	Á
58	:	84	T
59	;	85	U
6Ø	<	86	V
61	=	87	W
62	>	88	Á
63	?	89	Y
64	@	9Ø	Z
65	A	91	[
66	B	92	£
67	C	93	]
68	D	94	↑
69	E	95	←
7Ø	F	96	
71	G	97	♠
72	H	98	
73	I	99	
74	J	1ØØ	
75	K	1Ø1	
76	L	1Ø2	
77	M	1Ø3	

CODIGO	CARACTER	COGIDO	CARACTER
104		130	
105		131	
106		132	
107		133	f1
108		134	f3
109		135	f5
110		136	f7
111		137	f2
112		138	f4
113		139	f6
114		140	f8
115	♥	141	SHIFT RETURN
116		142	MAYUSCULAS
117		143	
118		144	BLK
119		145	↑ CRSR
120	♣	146	RVS OFF
121		147	CLR HOME
122	♦	148	INST DEL
123		149	
124		150	
125		151	
126	π	152	
127		153	
128		154	
129		155	

CODIGO	CARACTER	CODIGO	CARACTER
156	PUR	182	
157	← CRSR	183	
158	YEL	184	
159	CYN	185	
160	SPACE	186	
161		187	
162		188	
163		189	
164		190	
165		191	
166		192	
167		193	♠
168		194	
169		195	
170		196	
171		197	
172		198	
173		199	
174		200	
175		201	
176		202	
177		203	
178		204	
179		205	
180		206	
181		207	

COMMODORE (Continuación)

CODIGO	CARACTER		CODIGO	CARACTER
2Ø8			232	
2Ø9			233	
21Ø			234	
211			235	
212			236	
213			237	
214			238	
215			239	
216			24Ø	
217			241	
218			242	
219			243	
22Ø			244	
221			245	
222			246	
223			247	
224	SPACE		248	
225			249	
226			25Ø	
227			251	
228			252	
229			253	
23Ø			254	
231			255	

APPLE

CODIGO	CARACTER		CODIGO	CARACTER
ØØ	@		26	^Z
Ø1	^A		27	ESC
Ø2	^B		28	^\
Ø3	^C		29	^]
Ø4	^D		3Ø	^↑
Ø5	^E		31	^_
Ø6	^F		32	SPACE
Ø7	BELL		33	!
Ø8	^H		34	"
Ø9	^I		35	#
1Ø	^J		36	$
11	^K		37	%
12	^L		38	&
13	RETURN		39	'
14	^N		4Ø	(
15	^O		41	)
16	^P		42	*
17	^Q		43	+
18	^R		44	,
19	^S		45	-
2Ø	^T		46	.
21	^U		47	/
22	^V		48	Ø
23	^W		49	1
24	^X		5Ø	2
25	^Y		51	3

APPLE (Continuación)

CODIGO	CARACTER		CODIGO	CARACTER
54	4		78	N
53	5		79	O
54	6		8Ø	P
55	7		81	Q
56	8		82	R
57	9		83	S
58	:		84	T
59	;		85	U
6Ø	<		86	V
61	=		87	W
62	>		88	X
63	?		89	Y
64	@		9Ø	Z
65	A		91	[
66	A		92	\
67	C		93	]
68	D		94	^
69	E		95	—
7Ø	F		96	\
71	G		97	a
72	H		98	b
73	I		99	c
74	J		1ØØ	d
75	K		1Ø1	e
76	L		1Ø2	f
77	M		1Ø3	g

APPLE (Continuación)

CODIGO	CARACTER		CODIGO	CARACTER
1Ø4	h		117	u
1Ø5	i		118	v
1Ø6	j		119	w
1Ø7	k		12Ø	x
1Ø8	l		121	y
1Ø9	m		122	z
11Ø	n		123	{
111	o		124	|
112	p		125	}
113	q		126	~
114	r		127	DEL
115	s		129	@
116	t			

Los caracteres 128-255 son como los 0-127, pero en vídeo inverso.

APENDICE C

EQUIVALENCIAS EN CASTELLANO



Algunas versiones del LOGO presentan sus primitivas y palabras reservadas en castellano. Aunque no todas han elegido los mismos nombres, los más comunes se recogen en la tabla siguiente.

INGLES	CASTELLANO
.CALL	.LLAMA
AND	AMBOS, Y
ASCII	NUMCARACTER, ASCII
BACK	ATRAS, RETROCEDE
BG	FONDO, COLORFONDO
BUTFIRST	MENOSPRIMERO
BUTLAST	MENOSULTIMO
CATALOG	CATALOGO
CHAR	ESCRIBECARACTER, CARACTER
CLEAN	BORRA, LIMPIA
COS	COSENO
COUNT	CUENTA
CS	BORRAPANTALLA
EDIT	EDITAR, REVISAR
END	FIN

INGLES	CASTELLANO
ERALL	SUPTODO
ERASE	SUPRIME
ERF	ELIMINAARCHIVO
ERNS	SUPNS
ERPS	SUPPS
FALSE	FALSO
FIRST	PRIMERO
FORWARD	ADELANTE
HEADING	RUMBO
HOME	CENTRO
HT	OCULTATORTUGA
IF	SI
INT	ENT
LAST	ULTIMO
LEFT	IZQUIERDA, GIRAIZQUIERDA
LOAD	RECUERDA, CARGA
MAKE	HACER, ASIGNA
NODES	NODOS
NOT	NO
OR	UNOUOTRO, O
OUTPUT	RESPUESTA, DEVUELVE
PC	COLOR, COLORLAPIZ
PD	CONPLUMA, CONLAPIZ
PE	PLUMADEBORRAR, GOMA
PO	EI, ESCPROC
POALL	ET, ESCTODO
PONS	EN, ESCNS
POPS	ESCPROCS
POS	DONDE, POSICION
POTS	EP, ESCTS
PRINT	IMPRIME
PRODUCT	PRODUCTO
PU	SINPLUMA, SINLAPIZ
PX	PLUMAINVERSA, LAPIZREVES
QUOTIENT	COCIENTE
RANDOM	AZAR
RC	LEECARACTER, LEECAR
REMAINDER	RESTO
REPEAT	REPETIR, REPITA
RIGHT	DERECHA, GIRADERECHA
RUN	CUMPLE, EJECUTA
SAVE	GUARDA
SETBG	COLORFONDO, PONCF

INGLES	CASTELLANO
SETH	PONRUMBO
SETPC	PONCOLOR, PONCL
SETPOS	PXY, PONPOS
SETX	PX, PONX
SETY	PY, PONY
SHOW	MUESTRA
SIN	SENO
SQRT	RAIZCUADRADA
ST	MUESTRATORTUGA
STOP	PARATE, ALTO
SUM	SUMA
TELL	DECIR
TO	PARA
TRUE	CIERTO
TS	MODOTEXTO
TYPE	ESCRIBE
WHO	QUIEN
WRAP	ENROLLA
XCOR	XCOOR
YCOR	YCOOR

NOTAS

Uno de los lenguajes de programación más "de moda" en los últimos tiempos es, sin duda, el LOGO. Nació como consecuencia de diversas investigaciones sobre inteligencia artificial, pero ahora está disponible en versiones para los más conocidos y populares ordenadores domésticos y personales y su aplicación está muy extendida en escuelas y universidades, donde llega a codearse con el BASIC.

Fácil de asimilar y con un aprendizaje muy rápido el LOGO es, además, un lenguaje sumamente potente. Junto a una capacidad gráfica envidiable, que permite la realización de complejos dibujos, dispone de un manejo sencillo y claro de texto y cadenas de caracteres, además de peculiaridades tan interesantes como la recursividad y otras muchas que en este volumen de la B.B.I. les explicaremos en profundidad con una gran cantidad de probados ejemplos prácticos.

395 pts.
(incluido IVA)